An Adaptive Configuration-Aware Simulated Annealing for the Maximally Diverse Grouping Problem

Baiyu Chen, Canhui Luo, Junwen Ding, Qingyun Zhang*, Zhouxing Su, Zhipeng Lü

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
qingyun_zhang@hust.edu.cn

Abstract

The maximally diverse grouping problem (MDGP) seeks to partition the vertices of a complete graph into a fixed number of groups under capacity constraints, maximizing the sum of edge weights within each group. MDGP is an NP-hard combinatorial optimization problem and has wide real-world applications. In this paper, we propose an adaptive configuration-aware simulated annealing (ACSA) algorithm to solve MDGP. First, ACSA adopts a relaxation-based insertion strategy, which temporarily relaxes capacity constraints to expand the neighborhood and allow effective exploration of promising regions. Second, a memory-based swap mechanism is introduced to integrate high-potential suboptimal swap moves into the conventional best-swap operation, thereby achieving a better balance between diversification and intensification of the search. Finally, ACSA employs a vertex-wise sequential coordination strategy to dynamically organize the insertion and swap moves, which enhances the search flexibility. Experiments on 500 benchmark instances demonstrate the strong competitiveness of ACSA, as it improves the best results among the state-of-the-art algorithms on 460 instances and matches them on 39 instances.

Introduction

Given a complete edge-weighted undirected graph, the maximally diverse grouping problem (MDGP) aims to partition the vertices into a fixed number of groups, maximizing the sum of edge weights within each group under the capacity constraints. MDGP has been proven to be NP-hard (Feo and Khellaf 1990) and is widely used in various applications, such as storage allocation (Král 1965), VLSI design (Feo and Khellaf 1990; Li and Behjat 2006; Areibi et al. 2007), exam arrangement (Lotfi and Cerveny 1991; Weitz and Lakshminarayanan 1997), creation of peer review groups (Hettich and Pazzani 2006; Chen et al. 2011), and assignment of students to groups (Muller 1989; Krass and Ovchinnikov 2010; Yeoh and Mohamad Nor 2011; Johnes 2015; Schulz 2022).

The existing methods can be classified into two main categories: exact and metaheuristic methods. The most representative exact algorithm is the column generation algorithm proposed by Johnson, Mehrotra, and Nemhauser (1993). In fact, MDGP is challenging to solve both theoretically and practically. Gallego et al. (2013) attempted to solve MDGP by using commercial solver CPLEX, but failed to obtain the optimal solutions within an acceptable time for instances with more than 30 vertices. Therefore, heuristic algorithms remain the dominant approach in the literature to find high-quality suboptimal solutions for large instances. Early heuristic algorithms include a multi-start algorithm (Arani and Lotfi 1989), the Weitz-Jelassi (WJ) algorithm (Weitz and Jelassi 1992), which incorporates a heuristic for initial solution construction, and the Lotfi-Cerveny-Weitz (LCW) algorithm (Weitz and Lakshminarayanan 1998), which first constructs an initial solution and then applies a local search procedure.

With the development of metaheuristic algorithms, various advanced diversification strategies have been proposed to enhance local search. Among heuristic methods, variable neighborhood search has been widely adopted. For example, Palubeckis, Karčiauskas, and Riškus (2011) introduced the basic variable neighborhood search algorithm (VNS) for MDGP, which was later extended into a general variable neighborhood search (GVNS) by Uroševic (2014). Based on this, Brimberg, Mladenović, and Urošević (2015) proposed a skewed general variable neighborhood search (SGVNS) to further enhance GVNS. Besides, Palubeckis, Karčiauskas, and Riškus (2011) proposed a multi-start simulated annealing (MSA) and Gallego et al. (2013) developed a tabu search with strategic oscillation (TS-SO). In addition to trajectory-based search methods, population-based hybrid algorithms have also been extensively studied, such as the hybrid genetic algorithm (LSGA) (Fan et al. 2011), the hybrid grouping genetic algorithm (Chen et al. 2011), the hybrid steady-state genetic algorithm (HGA) (Palubeckis, Karčiauskas, and Riškus 2011), and the artificial bee colony algorithm (ABC) (Rodriguez et al. 2013).

Apart from the above works, impressive results were obtained by several recent metaheuristic algorithms. Palubeckis, Ostreika, and Rubliauskas (2015) proposed an iterated tabu search algorithm (ITS) by combining pairwise interchanges and relocations via a tabu strategy. Lai and Hao (2016) developed an iterated maxima search (IMS) which introduces weak and strong perturbations. Singh and Sundar (2019) proposed a novel hybrid genetic algorithm (NSGGA), which integrates a steady-state genetic frame-

*Corresponding author

work with a local search heuristic, specifically designed for instances where all groups have equal sizes. Lai et al. (2021) applied a novel neighborhood decomposition-based heuristic algorithm (NDHA) that combines tabu search and variable neighborhood descent. Yang et al. (2022) proposed a three-phase search approach with dynamic population size (TPSDP), which consists of three stages: undirected perturbation, restructure phase, and directed perturbation. Wu et al. (2025) introduced the feasible and infeasible region (FIFR) algorithm, which integrates infeasible region search with a genetic algorithm framework.

Although previous algorithms such as FIFR, TPSDP, and NDHA have made great progress in solving MDGP, there is still room for improvement. In this paper, we propose a new algorithm called adaptive configuration-aware simulated annealing algorithm (ACSA) for solving MDGP.

Our main contributions are summarized as follows:

(1) We adopt a relaxation-based insertion strategy that temporarily allows capacity violations and resolves them by redistributing the capacity constraints between the two involved groups. This approach overcomes the structural limitations of traditional insertion, effectively expanding the neighborhood and enhancing the exploration capability.

(2) We propose a memory-based swap mechanism that selectively reuses the previously retained best swap candidates via a configuration-based validity check strategy. By dynamically exploiting these slightly degraded suboptimal candidates, our method strikes an effective balance between diversification and intensification of the search.

(3) We introduce a vertex-wise sequential coordination strategy for traditional simulated annealing, forming an effective search procedure called configuration-aware simulated annealing (CSA). This strategy integrates the proposed insertion and swap operations at the vertex level, enabling a real-time response to structural changes and improving the search flexibility.

(4) Experiments on 500 instances demonstrate the strong competitiveness of ACSA, as it improves the best objective values obtained by the state-of-the-art algorithms on 460 instances and matches the same values on 39 instances.

Preliminaries

Problem Description

Given a complete edge-weighted undirected graph G, the set of N vertices is denoted as $V = \{v_1, v_2, \ldots, v_n\}$ and $w_{i,j} = w_{j,i}$ ($w_{i,j} \in \mathbb{R}_0^+$, the weights are non-negative values) is the weight of the edge between vertices v_i and v_j. The maximally diverse grouping problem aims to find a partition of V into K (K is specified) groups $\pi = \{C_1, \ldots, C_K\}$, and the size of each group $|C_i|$ ($1 \leq i \leq K$) must lie between the lower bound L_i and the upper bound U_i. Mathematically, the constraints of MDGP can be formulated as follows:

$$C_i \cap C_j = \varnothing,\ \forall i \neq j \tag{1}$$

$$\bigcup_{i=1}^{K} C_i = V \tag{2}$$

$$L_i \leq |C_i| \leq U_i,\ i \in \{1, 2, \ldots, K\} \tag{3}$$

The objective of MDGP is to maximize the sum of edge weights over all the groups, which can be formulated as follows:

$$\max f(\pi) = \sum_{i=1}^{K} \sum_{\substack{u,v \in C_i \\ u \neq v}} w_{u,v} \tag{4}$$

Important Symbols of ACSA

Some important notations in ACSA are defined as follows:

- C_i: the i-th group in the solution.
- C_{old} and C_{new}: the original and destination group of a vertex during a move.
- $|C_i|$: the size of group C_i.
- $\gamma(v)$: the group number that vertex v belongs to.
- π: the current solution.
- π^*: the best solution found so far.
- L_i and U_i: the lower and upper bounds of group C_i.
- P_{init}: the preset similarity threshold of two solutions.
- P_{final}: the preset percentage of stable vertices.
- T_{init} and T_{final}: the initial and final temperatures.

ACSA Algorithm

General Framework of ACSA

A distinctive feature of ACSA is the adoption of two novel neighborhood moves: The relaxation-based insertion (RBI) move is designed to effectively expand the neighborhood by temporarily relaxing capacity constraints, while the memory-based swap (MBS) move is tailored to quickly identify the swap partner with high exploratory potential for a selected vertex. Moreover, different from previous stage-wise local search approaches, such as FIFR, TPSDP, and NDHA, which divide insertion and swap moves into separate phases, our algorithm sequentially alternates between the two proposed neighborhood moves in a vertex-wise manner. This design promotes immediate coordination between insertion and swap moves, thereby establishing a more flexible search process.

The general framework of ACSA is presented in Algorithm 1. First, ACSA performs the initialization phase (lines 1-3). It randomly generates an initial solution π (line 1). Next, the initial and final temperatures (T_{init} and T_{final}) are determined by two target percentages P_{init} and P_{final} (line 3). Then, ACSA performs the search procedure until the elapsed time exceeds $time_{limit}$ (lines 4-18).

At each iteration, ACSA first sets the current temperature T to T_{init} (line 5), and then applies the configuration-aware simulated annealing (CSA) to improve the current solution π (lines 7-17). Similar to traditional simulated annealing, CSA has two layers of loops: an outer loop governed by the final temperature T_{final} (line 7), and an inner loop with a maximum number of iterations I_{in} (line 9). Before each inner loop, $vSwap$ and $toGroup$ are cleared and redefined to record the previous best swap partner and target group of each vertex (line 8), respectively. In each inner iteration, a vertex v is randomly selected from V (line 10). The algorithm prioritizes performing a relaxation-based insertion on

Algorithm 1: The main framework of the ACSA algorithm

Input: An MDGP instance
Output: The best solution π^* found so far
1: $\pi \leftarrow$ Generate an initial solution
2: $\pi^* \leftarrow \pi$
3: (T_{init}, T_{final}) $\leftarrow$Based on two target percentages (P_{init} and P_{final}), the corresponding temperatures are obtained through a bisection method
4: **while** elapsed time does not exceed $time_{limit}$ **do**
5: $T \leftarrow T_{init}$
6: $I_{in} \leftarrow N \cdot \theta$ /* The inner loop cutoff condition */
7: **while** $T \geq T_{final}$ **do** /* Begin CSA */
8: $vSwap$, $toGroup \leftarrow$ Array of size N to be with value -1
9: **for** $i = 0 \rightarrow I_{in}$ **do**
10: $v \leftarrow$ randInt$[1, N]$
11: $\pi \leftarrow$ RBI$(v, T, vSwap, \pi)$
12: Update π^* to π if $f(\pi) > f(\pi^*)$
13: $\pi \leftarrow$ MBS$(v, T, vSwap, toGroup, \pi)$
14: Update π^* to π if $f(\pi) > f(\pi^*)$
15: **end for**
16: $T \leftarrow T \cdot \lambda$
17: **end while**
18: **end while**
19: **return** π^*

vertex v (line 11). Once the insertion move is completed, the algorithm switches to the memory-based swap procedure (line 13). If the new solution π yields a better objective value than the current best solution π^*, CSA updates π^* accordingly (lines 12 and 14). After each inner loop, the temperature decreases by a cooling factor λ (line 16).

Initialization

The initialization phase involves both the generation of an initial solution and the determination of the initial and final temperatures. The initial solution generation consists of two steps. In the first step, vertices are randomly assigned to groups whose sizes have not yet reached their lower bounds. In the second step, the algorithm prioritizes filling the groups that have the largest upper bounds, and repeats this process until all vertices are assigned.

The initial and final temperatures are key parameters in simulated annealing. The initial temperature should enable sufficient exploration during the search, while the final temperature should guarantee the convergence of the search. We propose two methods to evaluate the effectiveness of these temperature settings. For the initial temperature, our algorithm first performs a brief search at the given temperature to obtain a warmed-up solution. It then continues the search until termination, and finally calculates the similarity between the warm-up solution and the final solution based on Eqs. (5) and (6). If the similarity is below P_{init}, it indicates that the search at this temperature demonstrates effective exploratory behavior. For the final temperature, our algorithm performs periodic sampling at the given temperature, recording the proportion of stable vertices (a stable vertex means that any move of that vertex will lead to a worse objective value) for each sampled solution and then calculating the average proportion. If the average value exceeds P_{final}, the search is deemed to be converged.

$$I(A, B) = \{x | x \in A \cap B\} \tag{5}$$

$$rate_{sim} = \frac{1}{2 \cdot N} \cdot \Big(\sum_{A_i \in \pi_1} max\{|I(A_i\ ,\ B)|\ , B \in \pi_2\} + \sum_{A_i \in \pi_2} max\{|I(A_i\ ,\ B)|\ , B \in \pi_1\} \Big) \tag{6}$$

Subsequently, we use a binary search to find a suitable T_{init} (T_{final}) in the range $[0.01, 2000]$ for a given instance. For this purpose, we run CSA where both initial and final temperatures are set to the middle value in the initial range $[0.01, 2000]$. If this leads to a percentage close to P_{init} (P_{final}), then T_{init} (T_{final}) is definitively set to this value. Otherwise, we re-run CSA with the first or second half-sized range according to whether the percentage is lower or higher than P_{init} (P_{final}).

Neighborhood Structure

Neighborhood structure plays an important role in local search-based metaheuristics. Two widely used operations for MDGP are insertion and swap: Insertion moves a vertex from one group to another, while swap exchanges two vertices between different groups. As the fast neighborhood evaluation technique has proven successful in previous algorithms (Aringhieri and Cordone 2011; Palubeckis, Karčiauskas, and Rišķus 2011), we adopt the same method. Since the number of partitions K is fixed in advance, we define an auxiliary $K \times N$ array W, where $W_{C_k,v}$ denotes the total edge weight between vertex v and the vertices in C_k:

$$W_{C_k,v} = \sum_{\substack{u \in C_k \\ v \neq u}} w_{v,u} \tag{7}$$

When moving vertex v from the current group C_{old} to another group C_{new}, the objective gain can be calculated as follows:

$$\Delta = W_{C_{new},v} - W_{C_{old},v} \tag{8}$$

The objective gain of the swap between vertices v and u is calculated in a similar way:

$$\Delta = W_{C_{\gamma(u)},v} - W_{C_{\gamma(v)},v} + W_{C_{\gamma(v)},u} - W_{C_{\gamma(u)},u} - 2w_{v,u} \tag{9}$$

Our algorithm adopts a best-improvement strategy, in which the best target group for the insertion move and the best swap partner for the swap move are selected for a given vertex, respectively. It can be observed that evaluating an insertion move has a time complexity of $O(K)$, while evaluating a swap move has a time complexity of $O(N)$. When vertex v is moved from C_{old} to C_{new}, $W_{C_{old},i}$ and $W_{C_{new},i}$ need to be updated:

$$W_{C_{old},i} = W_{C_{old},i} - w_{i,v} \ \ (1 \leq i \leq N) \tag{10}$$

$$W_{C_{new},i} = W_{C_{new},i} + w_{i,v} \ \ (1 \leq i \leq N) \tag{11}$$

A swap move involves relocating two vertices and can thus be approximated as two insertion moves. As a result, the time complexity of updating W remains $O(N)$ for both insertion and swap moves.

Configuration-Aware Simulated Annealing

In traditional double-neighborhood search methods, such as FIFR, TPSDP, and NDHA, insertion and swap moves are typically organized into two distinct phases that are repeatedly alternated during the search process. The insertion phase first adjusts group sizes by relocating vertices, and then the swap phase refines vertex assignments under the fixed group sizes. Despite its good performance, this stage-based design introduces a form of delayed coordination. Many potential improvements rely on the immediate coordination between insertion and swap moves, but the temporal separation of the two moves, often spanning many iterations, prevents the timely execution of such coordinated moves. As a result, the opportunities for synergistic optimization are often missed, limiting the deep exploration of the solution space and reducing the chance of discovering high-quality solutions.

To address the limitations of stage-wise neighborhood design, we propose a configuration-aware simulated annealing (CSA), which is outlined in Algorithm 1. The core of CSA lies in a vertex-wise sequential coordination strategy, in which insertion and swap operations are sequentially attempted for a randomly selected vertex. Specifically, an insertion move is first applied to explore potential changes in group sizes, followed immediately by a swap move to exploit structural adjustments induced by the insertion move. This tight coupling allows CSA to dynamically sense and respond to changes in solution configuration (the group size distribution within the solution) at each step. Through such micro-level continuous coordination, a macro-level synergy emerges: The insertion move expands the search space for swap, and the swap move, in turn, exploits the potential improvements introduced by insertion. This timely coordination effectively mitigates the loss of improvement opportunities resulting from delays in stage-wise processing, enhancing overall search efficiency.

Relaxation-based Insertion

We propose a relaxation-based insertion strategy to overcome the neighborhood restrictions caused by capacity constraints. Due to capacity constraints, the traditional insertion move often fails to effectively implement the best-improvement strategy, which always moves a vertex to the group with the maximum objective gain. Specifically, on the one hand, inserting a vertex into a full group will break the upper bound constraint; on the other hand, removing a vertex from a group that is exactly at the lower bound will also be infeasible. However, some insertion moves are mistakenly considered infeasible in the traditional approach. For example, as illustrated in Figure 1, when the sizes of groups A and B are exactly equal to the lower and upper bounds of the group capacity constraints, respectively, moving a vertex from group A to group B is considered a constraint violation, even though the overall capacity requirements are still satisfied. This is due to the restrictive binding of capacity constraints to individual groups, which prevents flexible redistribution of constraints during the search.

To improve the conventional insertion move, the relaxation-based insertion introduces a two-stage capacity feasibility check. First, the insertion move is feasible if both involved groups satisfy their original capacity constraints after the insertion:

$$\begin{cases} L_{new} \leq |C_{new}| + 1 \leq U_{new} \\ L_{old} \leq |C_{old}| - 1 \leq U_{old} \end{cases} \tag{12}$$

Otherwise, the fixed capacity bounds assigned to each group are relaxed by interchanging the upper and lower bounds between the two groups. The insertion move remains valid as long as the resulting sizes of both involved groups satisfy the updated capacity constraints and the move obtains a positive objective gain:

$$\begin{cases} L_{old} \leq |C_{new}| + 1 \leq U_{old} \\ L_{new} \leq |C_{old}| - 1 \leq U_{new} \\ \Delta \geq 0 \end{cases} \tag{13}$$

By accepting previously infeasible but high-gain insertion moves, this strategy effectively expands the search space while guiding the search toward high-quality solutions.

Algorithm 2 presents the procedure of relaxation-based insertion. The algorithm evaluates all groups that satisfy either the standard constraint (line 5) or the relaxed constraint (line 7), and selects the one with the maximum objective

Algorithm 2: Relaxation-based Insertion (RBI)

Input: The selected vertex v, the temperature T, the recorded swap partners $vSwap$, the current solution π
Output: The current solution π

1: $\Delta_{max} \leftarrow -\infty, C_{new} \leftarrow \varnothing, C_{old} \leftarrow \gamma(v), type \leftarrow \varnothing$
2: **for** $i = 1 \rightarrow K$ **do**
3: Calculate the gain Δ of moving v to group C_i by Eq. (8)
4: **if** $\Delta_{max} < \Delta$ and $C_i \neq C_{old}$ **then**
5: **if** new solution satisfies Eq. (12) **then**
6: $\Delta_{max} \leftarrow \Delta, C_{new} \leftarrow C_i, type \leftarrow standard$
7: **else if** new solution satisfies Eq. (13) **then**
8: $\Delta_{max} \leftarrow \Delta, C_{new} \leftarrow C_i, type \leftarrow relaxed$
9: **end if**
10: **end if**
11: **end for**
12: **if** $C_{new} \neq \varnothing$ and randReal$(0, 1) \leq exp(\frac{\Delta_{max}}{T})$ **then**
13: $vSwap[v] \leftarrow -1$
14: $\pi \leftarrow \pi \oplus$ Move(v, C_{old}, C_{new})
15: Update W according to Eqs. (10) and (11)
16: **if** $type = relaxed$ **then**
17: Swap bounds of C_{new} and C_{old}
18: **end if**
19: **end if**

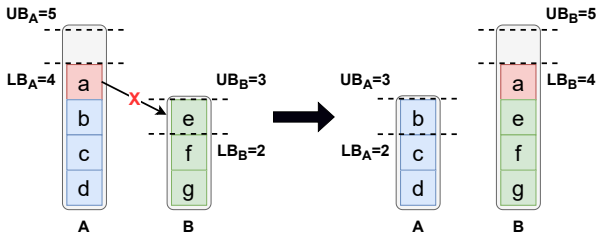

Figure 1: Illustration of the relaxation-based insertion.

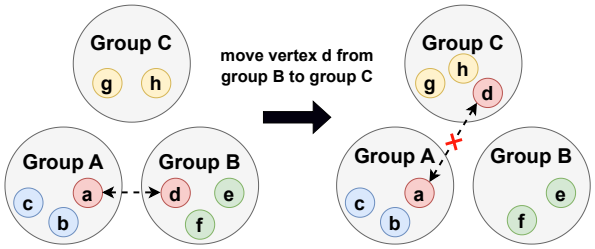

Figure 2: Illustration of the configuration-based validity check strategy.

gain (lines 2-11). The $type$ is updated according to which constraint is satisfied (lines 6 and 8). If a valid target group is found and the corresponding move is accepted, the cached swap partner of vertex v in $vSwap$ is cleared (line 13) and the solution is updated (line 14). Additionally, if the move is performed under the relaxed constraint, the upper and lower bounds of the two involved groups are exchanged accordingly (line 17).

Memory-based Swap

ACSA adopts a memory-based swap mechanism that selectively reuses cached swap candidates to improve the traditional best-swap move. For a given vertex, the conventional best-swap move always selects the cross-group swap partner with the highest objective gain and performs the swap accordingly. Although the best-swap move is generally effective, it tends to drive the search towards premature convergence, potentially falling into local optima traps. Furthermore, its exhaustive cross-group candidate evaluations incur considerable computational overhead for large instances.

To enhance search diversification and reduce evaluation overhead, the memory-based swap mechanism strategically reuses suboptimal candidates that were previously recorded as the best swap partners, but may have degraded due to changes in the solution during the search. The key challenge is determining whether these cached candidates remain sufficiently effective. Directly comparing the cached swap partner with the current best one for a given vertex incurs costly re-evaluation and lacks a guiding scoring function. Instead of relying on this direct comparison, we exploit the inherent connection between the best swap move and swap configuration (the current composition of vertices within the two groups involved in the swap). Although the best-swap evaluation is performed at the vertex level, it essentially reflects a group-level preference: The initiating vertex favors moving to a particular group under a specific swap configuration.

This fact motivates us to propose a configuration-based validity check strategy that reuses the cached swap partner of a selected vertex only when the vertex has a recorded swap partner and the partner remains in the same group as in the last evaluation; otherwise, the best swap partner is re-evaluated. Moreover, once any vertex is moved, its cached swap partner will be cleared. Figure 2 illustrates how changes in the swap configuration can invalidate the cached swap information. Since the quality of a swap depends more on the swap configuration of the two involved vertices than on the vertices themselves, moving either vertex of the swap

Algorithm 3: Memory-based Swap (MBS)

Input: The selected vertex v, the current temperature T, the recorded swap partners $vSwap$, the recorded target groups $toGroup$, the current solution π
Output: The current solution π

1: $u \leftarrow vSwap[v]$, $\Delta \leftarrow -\infty$
2: **if** $u = -1$ or $toGroup[v] \neq \gamma(u)$ **then**
3: $u, \Delta \leftarrow$ Find the best swap partner and gain Δ for vertex v from other groups according to Eq. (9)
4: $vSwap[v] \leftarrow u$, $toGroup[v] \leftarrow \gamma(u)$
5: **else**
6: $\Delta \leftarrow$ Calculate the gain Δ of swapping vertices u and v according to Eq. (9)
7: **end if**
8: **if** randReal$(0, 1) \leq exp(\frac{\Delta}{T})$ **then**
9: $vSwap[v] \leftarrow -1$, $vSwap[u] \leftarrow -1$
10: $\pi \leftarrow \pi \oplus$ Swap(v, u)
11: Update W according to Eqs. (10) and (11)
12: **end if**

is likely to alter their swap configuration, potentially rendering the cached information invalid and necessitating a re-evaluation. Through the configuration-based validity check strategy, the memory-based swap mechanism strategically tolerates slight deterioration of cached swap partners to promote exploration, and decisively rejects sharply degraded swap partners to preserve solution quality, striking an effective balance between diversification and intensification of the search. In addition, by reusing the cached swap candidates, this mechanism significantly reduces the frequency of evaluation, improving the search efficiency.

The process of the memory-based swap is described in Algorithm 3. The algorithm first obtains the cached swap partner u for vertex v (line 1). If the cache is invalid, which means that no swap partner is stored or the current group of the cached partner u differs from the previously recorded target group (line 2), the algorithm evaluates all vertices that are not in the same group as v to find the best swap partner u along with the corresponding gain (line 3). The cache is then updated by setting $vSwap[v]$ to u and $toGroup[v]$ to $\gamma(u)$ (line 4). Otherwise, the gain of swapping v and u is directly calculated (line 6). If the swap move is accepted, the cached entries of both v and u in $vGroup$ are cleared (line 9) and the solution π is updated (line 10). Finally, the auxiliary data structure W is updated (line 11).

Computational Results

Experimental Protocol and Benchmarks

Our ACSA[1] algorithm is implemented in C++ and runs on an Intel (R) Xeon (R) Gold 6133 processor with 2.50 GHz CPU and 128 GB RAM. We use the same benchmark instances[2] as those widely adopted by previous MDGP studies

[1]The source code and detailed experimental results can be found at https://github.com/CBYer/MDGP

[2]Available at https://grafo.etsii.urjc.es/optsicom/mdgp/ and https://grafo.etsii.urjc.es/optsicom/mdg/

Vertices	Cutoff time (s)	Vertices	Cutoff time (s)
$n = 120$	3	$n = 240$	20
$n = 480$	120	$n = 960$	600
$n = 2000$	1200	$n = 3000$	3000

Table 1: Cutoff times in seconds for different instance sizes.

Param	Description	Candidate values	Final
P_{init}	Similarity threshold	{0.1, 0.2, 0.3, 0.4, 0.5}	0.4
P_{final}	Percentage of stable vertices	{0.96, 0.97, 0.98, 0.99, 1}	0.99
θ	Coefficient for the inner loop	{50, 100, 150, 200}	100
λ	Cooling ratio of CSA	{0.96, 0.97, 0.98, 0.99}	0.98

Table 2: Parameter tuning results.

(Yang et al. 2022; Wu et al. 2025). These instances include three small-scale sets and two large-scale sets, covering a total of 500 instances with sizes ranging from 120 to 3000. The cutoff times in seconds of the tested algorithms are listed in Table 1. Each benchmark set can be divided into two categories: instances with equal group sizes (EGS) and instances with different group sizes (DGS). Next, the characteristics of each benchmark set are described in detail.

Each of the RanInt, RanReal, and Geo sets consists of 80 instances, covering four problem sizes with 120, 240, 480, and 960 vertices. For each size, there are 10 EGS and 10 DGS instances, with a total of 20 instances. RanInt set uses integer edge weights uniformly drawn from (0,100), while the RanReal set uses real-valued weights in the same range. The number of groups K ranges from 10 to 24, with group size bounds between 2 and 48. The Geo set follows the same structural settings but defines edge weights as Euclidean distances between randomly generated coordinates in dimensions from 2 to 21. The MDG-a set comprises 220 instances, derived from 20 base graphs with 2000 vertices, each of which is tested under 11 different capacity constraint configurations. The edge weights in these instances are uniformly drawn as integers from [0,10]. The MDG-c set includes 40 instances with 3000 vertices and edge weights drawn from [0,1000]. These instances are tested under both EGS and DGS constraints with a K value of 50.

Parameter Tuning

We use the automatic configuration tool "irace" (López-Ibáñez et al. 2016) to calibrate the parameters of the ACSA algorithm. For this purpose, we run irace on 28 randomly selected instances of different scales with a total tuning budget of 5600 runs. We adopt the best parameter configuration given by irace as shown in Table 2.

Comparison with the State-of-the-Art Algorithms

To the best of our knowledge, there are three recent best-performing MDGP algorithms: FIFR (Wu et al. 2025), TPSDP (Yang et al. 2022), and NDHA (Lai et al. 2021). We apply ACSA, FIFR, TPSDP, and NDHA to each instance with 20 independent runs under the same environment. The cutoff times of the tested algorithms listed in Table 1 are the same as those in previous reference algorithms.

Type	Instance	ACSA		FIFR		TPSDP		NDHA	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg
DGS	Geo	**39**	**35**	0	4	0	0	1	1
	RanReal	**40**	**40**	4	0	1	0	2	0
	RanInt	**40**	**40**	1	0	0	0	1	0
	MDG-a	**120**	**120**	0	0	0	0	0	0
	MDG-c	**20**	**20**	0	0	0	0	0	0
EGS	Geo	**40**	**40**	0	0	0	0	0	0
	RanReal	**40**	**40**	13	0	16	0	9	0
	RanInt	**40**	**40**	15	2	14	1	10	0
	MDG-a	**100**	**100**	0	0	0	0	0	0
	MDG-c	**20**	**20**	0	0	0	0	0	0
NoB	All	**499**	**495**	33	6	31	1	23	1

Table 3: Summary of comparison among ACSA, FIFR, TPSDP, and NDHA on all the tested instances.

Instance	Algorithm	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
DGS (260)	FIFR	254	1	1.8E-43	255	5	3.2E-43
	TPSDP	258	1	3.2E-44	255	5	1.3E-43
	NDHA	256	1	9.5E-44	257	3	3.9E-44
EGS (240)	FIFR	212	0	1.5E-36	238	0	8.5E-41
	TPSDP	210	0	3.2E-36	239	0	5.8E-41
	NDHA	221	0	5.1E-38	240	0	4.0E-41
Total (500)	Top baseline	460	1	4.6E-77	493	5	1.3E-81

Table 4: Wilcoxon signed ranks test results of ACSA and the reference algorithms in terms of both the best and the average solutions on 500 instances, with a significance level of 0.05.

Table 3 reports the number of instances on which ACSA, FIFR, TPSDP, and NDHA achieve the best performance in terms of the best and average objective values across all tested instances. The row "NoB" summarizes the total number of instances where each algorithm achieves the best results. Table 4 presents the comparison of ACSA with FIFR, TPSDP, and NDHA in terms of the average and best objective values, respectively. R^+_{best} and R^+_{avg} record the number of times for which ACSA outperforms the reference algorithms in terms of the best and average objective values, respectively, while R^-_{best} and R^-_{avg} represent the number of times for which ACSA performs worse than the reference algorithms in terms of the best and average objective values. The row "Total (500)" summarizes the comparison between ACSA and the best-performing result among the three baseline algorithms (FIFR, TPSDP, and NDHA) for each instance.

From Table 3, one observes that ACSA significantly outperforms other reference algorithms, as it obtains the best results for the largest number of times in terms of both the best and average objective values. Specifically, out of the 500 tested instances, ACSA achieves the best objective value in 499 cases and the best average performance in 495 cases. In addition, ACSA demonstrates good performance across both DGS and EGS instance types, highlighting its robustness and general applicability. According to Table 4, ACSA achieves better best objective values than the reference al-

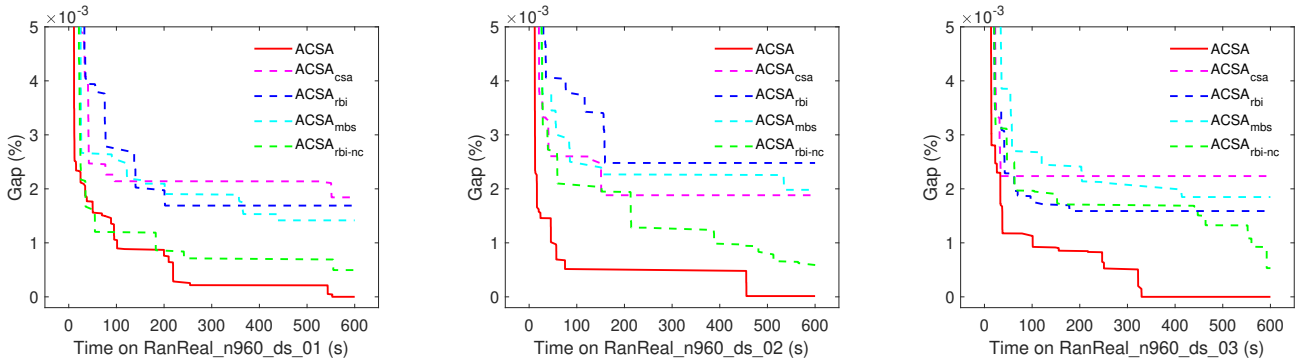

Figure 3: Evolution of objective values by ACSA and its four variants on three representative instances.

Comparison	R^+_{best}	R^-_{best}	R^+_{avg}	R^-_{avg}
ACSA$_{csa}$	88	16	101	19
ACSA$_{rbi}$	100	18	106	14
ACSA$_{mbs}$	89	8	112	8
ACSA$_{rbi-nc}$	54	41	76	44

Table 5: Comparison of ACSA and its four variants on 120 representative instances.

gorithms on 460 instances. Finally, as presented in Table 4, all p-values are well below 0.05, confirming the statistically significant differences between ACSA and each of the other three compared algorithms.

Discussion and Analysis

In this section, we present an analysis of each important component in ACSA, including the vertex-wise sequential coordination strategy, the relaxation-based insertion strategy, and the memory-based swap mechanism. We consider four variants of ACSA, each disabling a different component and keeping other components unchanged.

- **ACSA**$_{csa}$: The insertion and swap moves are separated into two phases, which are performed sequentially in simulated annealing.
- **ACSA**$_{rbi}$: Replace the relaxation-based insertion with a conventional best insertion.
- **ACSA**$_{mbs}$: Replace the memory-based swap with a conventional best swap.
- **ACSA**$_{rbi-nc}$: Remove the constraint of requiring positive objective improvement in relaxation-based insertion.

We carried out experiments on 120 DGS instances from the Geo, RanInt, and RanReal datasets, all of which involve highly complex capacity constraints. Rather than using uniform bounds, each container imposes individual lower and upper bounds. The running settings of these variants are the same as those of ACSA. Table 5 presents the comparison of each variant with ACSA. R^+_{best} and R^+_{avg} (R^-_{best} and R^-_{avg}) record the number of times for which ACSA obtains better (worse) results than other variants in terms of the best and average objective values, respectively.

From Table 5, we can observe that ACSA outperforms other variants, because ACSA obtains better results with the largest number of times in terms of both the best and average objective values, and its hitting rate is much greater than that of other variants. This indicates that each strategy in ACSA is important. To further compare each variant, Figure 3 depicts the trend of the objective value on three representative instances (RanReal_n960_ds_01, RanReal_n960_ds_02, RanReal_n960_ds_03) when applying ACSA and other variants. Due to the large value of the objective function, each point (x, y) on the curve represents the gap between the objective value of the current solution and the best solution found by ACSA.

From Figure 3, one observes that ACSA always achieves the best performance across all tested variants. Specifically, during the convergence process, ACSA consistently outperforms ACSA$_{csa}$, ACSA$_{rbi}$, and ACSA$_{mbs}$. Notably, ACSA$_{csa}$ and ACSA$_{rbi}$ exhibit the worst overall performance. The reason that ACSA outperforms ACSA$_{csa}$ might be that the vertex-wise sequential coordination strategy enhances the flexibility and adaptability of the search process by dynamically organizing the insertion and swap operations. In the case of ACSA$_{rbi}$, its restricted neighborhood limits the search space compared to ACSA, which may reduce the potential for further improvement in solution quality. In addition, compared to ACSA$_{mbs}$, the memory-based swap mechanism in ACSA leverages cached suboptimal candidates to promote search diversification and reduce the evaluation frequency, thus improving search efficiency. Among all variants, ACSA$_{rbi-nc}$ is the closest to ACSA in performance. However, a notable gap remains. For ACSA$_{rbi-nc}$, the constraint on the positive objective improvement prevents frequent group-size changes, enabling deeper exploration under a relatively stable group structure.

Conclusions

We proposed an adaptive configuration-aware simulated annealing called ACSA for solving MDGP. By relaxing the capacity constraints, the relaxation-based insertion effectively expands the neighborhood search space. In addition, the diversification of ACSA is enhanced by the memory-based swap mechanism. Finally, ACSA integrates the two novel neighborhood moves via a vertex-wise sequential coordination strategy, which improves search flexibility. Computational experiments show the superior performance of ACSA in terms of both solution quality and search efficiency.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments. This work was supported in part by the Quantum Science and Technology-National Science and Technology Major Project under Grant No. 2024ZD0300500, the National Natural Science Foundation of China (NSFC) under Grant 6240219, 62202192 and 62572210, the Interdiciplinary Research Program of Hust 2025JCYJ021, and Interdisciplinary Research Program of Hust under Grant 5003300129.

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.

Brimberg, J.; Mladenović, N.; and Urošević, D. 2015. Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Information Sciences*, 295: 650 – 675.

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

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

Feo, T. A.; and Khellaf, M. 1990. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20(2): 181 – 195.

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.

Hettich, S.; and Pazzani, M. J. 2006. Mining for proposal reviewers: lessons learned at the national science foundation. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, 862–871.

Johnes, J. 2015. Operational research in education. *European Journal of Operational Research*, 243(3): 683 – 696.

Johnson, E. L.; Mehrotra, A.; and Nemhauser, G. L. 1993. Min-cut clustering. *Mathematical Programming*, 62(1-3): 133 – 151.

Krass, D.; and Ovchinnikov, A. 2010. Constrained group balancing: Why does it work. *European Journal of Operational Research*, 206(1): 144 – 154.

Král, J. 1965. To the problem of segmentation of a program. *Information Processing Machines*, 2.

Lai, X.; and Hao, J. 2016. Iterated maxima search for the maximally diverse grouping problem. *European Journal of Operational Research*, 254(3): 780–800.

Lai, X.; Hao, J.-K.; Fu, Z.-H.; and Yue, D. 2021. Neighborhood decomposition based variable neighborhood search and tabu search for maximally diverse grouping. *European Journal of Operational Research*, 289(3): 1067–1086.

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.

López-Ibáñez, M.; Dubois-Lacoste, J.; Pérez Cáceres, L.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3: 43–58.

Muller, T. E. 1989. Assigning students to groups for class projects: an exploratory test of two methods. *Decision Sciences*, 20(3): 623 – 634.

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.

Palubeckis, G.; Ostreika, A.; and Rubliauskas, D. 2015. Maximally diverse grouping: an iterated tabu search approach. *Journal of the Operational Research Society*, 66(4): 579–592.

Rodriguez, F. J.; Lozano, M.; García-Martínez, C.; and González-Barrera, J. D. 2013. An artificial bee colony algorithm for the maximally diverse grouping problem. *Information Sciences*, 230: 183 – 196.

Schulz, A. 2022. A new mixed-integer programming formulation for the maximally diverse grouping problem with attribute values. *Annals of Operations Research*, 318(1): 501 – 530.

Singh, K.; and Sundar, S. 2019. A new hybrid genetic algorithm for the maximally diverse grouping problem. *International Journal of Machine Learning and Cybernetics*, 10(10): 2921–2940.

Uroševic, D. 2014. Variable neighborhood search for maximum diverse grouping problem. *Yugoslav Journal of Operations Research*, 24(1): 21 – 33.

Weitz, R.; 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.

Weitz, R. R.; and Jelassi, M. T. 1992. Assigning students to groups: a multi-criteria decision support system approach. *Decision Sciences*, 23(3): 746 – 757.

Weitz, R. R.; 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.

Wu, X.; Feng, J.; Yang, J.; and Zhang, Y. 2025. Feasible and infeasible region search for the maximally diverse grouping problem. *Computers & Operations Research*, 179: 107030.

Yang, X.; Cai, Z.; Jin, T.; Tang, Z.; and Gao, S. 2022. A three-phase search approach with dynamic population size for solving the maximally diverse grouping problem. *European Journal of Operational Research*, 302(3): 925–953.

Yeoh, H. K.; and Mohamad Nor, M. I. 2011. An algorithm to form balanced and diverse groups of students. *Computer Applications in Engineering Education*, 19(3): 582 – 590.