An Elite-guided Weighted Simulated Annealing Algorithm for the Clique Partitioning Problem

Baiyu Chen, Junwen Ding, Canhui Luo, 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 clique partitioning problem (CPP) aims to find a partition of vertices of a complete graph in order to maximize the sum of edge weights within each partition (clique), which has been proven to be NP-hard and has wide real-world applications. In this paper, we propose an elite-guided weighted simulated annealing algorithm called EWSA to solve the CPP. First, EWSA employs two specific configurations and alternates between them via an oscillation strategy, which balances the exploitation and exploration of the search. Second, a weighting strategy is introduced to improve the scoring function in traditional simulated annealing, which is able to guide the search to explore diverse solutions. Finally, a partition restriction strategy is adopted to reduce search space and increase the search efficiency. Experiments on 255 instances demonstrate the competitiveness of EWSA. For 130 open instances, EWSA discovers new upper bounds in 32 cases and matches the best known results for the others. For the remaining 125 closed instances, EWSA achieves the best known objective values within a short computational time.

Introduction

Given a complete edge-weighted undirected graph, the clique partition problem (CPP) aims to find a partition of vertices of the graph such that the sum of edge weights that belong to each subset (clique) is maximized. The CPP has been proven to be NP-hard (Grötschel and Wakabayashi 1990) and was widely used in various applications, such as image processing (Ion, Carreira, and Sminchisescu 2011), social network analysis (Alduaiji, Datta, and Li 2018; Fox et al. 2020; Ouyang, Dey, and Zhang 2020), data mining (Miyauchi, Sonobe, and Sukegawa 2018), biological computing (Grötschel and Wakabayashi 1989; Strickland, Barnes, and Sokol 2005), and logistics (Dorndorf, Jaehn, and Pesch 2008; Wang et al. 2006).

Since the CPP was first introduced by Grötschel and Wakabayashi (1989), many methods have been proposed which can be classified into two main categories: exact and metaheuristic methods. The well-known exact approaches include: cutting plane (Grötschel and Wakabayashi 1989, 1990), branch-and-bound (Dorndorf and Pesch 1994), branch-and-price (Mehrotra and Trick 1998), and branch-and-price-and-cut (Ji and Mitchell 2007), etc. In addition, Jaehn and Pesch (2013) proposed new methods for constraining the upper bound of the CPP, while Sukegawa, Yamamoto, and Zhang (2013) reduced the problem size by Lagrange relaxation. Although exact algorithms are able to guarantee the optimality, they fail to solve large-scale instances in a reasonable time (Sørensen and Letchford 2024). Therefore, metaheuristics have become the popular methods for the CPP.

Various metaheuristic algorithms have been proposed for the CPP. Local search is the earliest metaheuristic algorithm applied to the CPP (Marcotorchino and Michaud 1981; Regnier 1983), which used the basic neighborhood move by reassigning a vertex to a subgraph, and iterates until reaching the local optima. With the development of metaheuristic algorithms, various advanced diversification strategies have been proposed to enhance the search capability of local search. Among them, tabu search (Glover 1989, 1990) and simulated annealing (Kirkpatrick, Jr., and Vecchi 1983; Skiscim and Golden 1983) are the most popular algorithms. Amorim, Barthélemy, and Ribeiro (1992) applied them to solve the CPP. More powerful algorithms were subsequently proposed for the CPP, such as noising methods (Charon and Hudry 2001, 2006), variable neighborhood search with embedded tabu search (Brusco and Koehn 2009) and others.

Apart from the above works, impressive results were obtained by several recent metaheuristic algorithms. In particular, Palubeckis, Ostreika, and Tomkevičius (2014) proposed an iterated tabu search algorithm (ITS) by combining the traditional tabu search with a perturbation procedure and introducing a neighborhood move which relocates two vertices in the same subgraph in the perturbation phase. Based on ITS, Zhou, Hao, and Goëffon (2016) developed an effective three-phase local search algorithm (CPP-P^3) which consists of three stages: descending search, tabu search, and directional perturbation. Hu et al. (2021) proposed a two-model local search algorithm with a self-adaptive parameter mechanism (TMLS-SA), which combines local search with simulated annealing and controls the perturbation intensity by means of an adaptive parameter during the search process. More recently, Lu, Zhou, and Hao (2022) applied a merge-divide memetic clique partitioning algorithm (MDMCP) to the CPP and produced very competitive results on instances

*Corresponding author

with up to 7,000 vertices. Jovanovic, Sanfilippo, and Voß (2023) and Gao et al. (2022) respectively proposed a fixed set search algorithm (FSS) and simulated annealing algorithm based on configuration checking (SACC). To the best of our knowledge, SACC and its variants are the best-performing metaheuristics up to now for solving the CPP.

Although previous algorithms such as SACC (Gao et al. 2022), FSS (Jovanovic, Sanfilippo, and Voß 2023), and MDMCP (Lu, Zhou, and Hao 2022) have made great progress in solving the CPP, there are still rooms for improvement for large-scale instances. In this paper, we propose a new algorithm called elite-guided weighted simulated annealing algorithm (EWSA) for solving the CPP.

Our main contributions can be summarized as follows:

(1) EWSA employs two search configurations: an elite-guided intensification configuration and an exploratory diversification configuration. In addition, EWSA oscillates between these two configurations via an oscillation strategy, which considers the quality of the current elite solution. This balances the exploitation and exploration of the search.

(2) We propose a weighting strategy for traditional simulated annealing, forming an effective search procedure called the weighted simulated annealing (WSA). The weighting strategy improves the scoring function in WSA by considering both the objective value and the vertex move frequency, which enhances the diversification of the search.

(3) EWSA adopts an adaptive partition number restriction strategy, which reduces the search space and improves the search efficiency.

(4) Experiments on 130 open instances show that EWSA outperforms the state-of-the-art algorithms since it improves and matches the previous best known results on 32 and 98 instances, respectively. For other 125 closed instances with known optimal solutions, EWSA matches all the optimal results in the literature.

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 $w_{i,j} = w_{j,i}$ ($w_{i,j} \in \mathbb{R}$, the weights contain negative values) is the weight of the edge between vertices v_i and v_j. The clique partitioning problem aims to find a partitioning of V into k (k is unspecified and needs to be determined by the algorithm) subsets (cliques) $\pi = \{C_1, \ldots, C_k\}$ satisfying the following constraints:

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

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

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

$$\max f(\pi) = \sum_{i=1}^{k} \sum_{\substack{u,v \in C_i \\ u \neq v}} w_{u,v} \tag{3}$$

Important Symbols of EWSA

Some important notations in EWSA are defined as follows:

- p_1 and p_2: two preset percentages of stable vertices.
- T_1 and T_2: two initial temperatures.
- T_{lb}: the lower bound of the final temperature.
- π: the current solution.
- π_{elite}: the elite solution found in recent iterations.
- π^*: the best solution found so far.
- $list_{elite}$: the set of elite solutions.
- sc: the scoring function selection indicator.
- $rate_{c2}$: the probability of using the second configuration.
- K_{pr}: the partition number restriction.
- $K(\pi)$: the number of partitions used by π.

EWSA Algorithm

General Framework of EWSA

A distinguishing feature of EWSA is the adaptive alternation between exploratory diversification configuration $config_1$ and an elite-guided intensification configuration $config_2$ during the search process. These two configurations endow EWSA with a better trade-off between intensification and diversification: $config_1$ is designed to discover better elite solutions, while $config_2$ is tailored to refine the current elite solution. Based on the differences between the two configurations, an effective and customized scoring function can be devised for each configuration by a weighting strategy.

The general framework of EWSA is presented in Algorithm 1. First, EWSA performs the initialization phase (lines 1-5). It greedily generates an initial solution π (line 1). The initial temperature T_1 and T_2 of WSA are obtained by different target percentages p_1 and p_2 (line 3). After performing a round of WSA under $config_2$, the temperature at which the best solution is last updated from T_2 to T_{lb} is recorded as T_{final} (line 4). Then, EWSA performs the search procedures until the elapsed time exceeds $time_{limit}$ (lines 6-20).

At each iteration of the search procedure, EWSA selects the search configuration based on the rate $rate_{c2}$ (lines 7-12). The two configurations primarily include the initial solution π for this round of WSA, the initial temperature T_{init}, and the scoring function indicator sc. Before each round of WSA, the adaptive partition restriction strategy sets the maximum number of partitions K_{pr} to be the number of partitions in the historical best solution found by EWSA, plus an empty partition (line 13). EWSA ensures that all vertices are in K_{pr} partitions by splitting the smallest partition and randomly reassigning its vertices to other partitions until the solution becomes valid (line 14). Afterwards, EWSA proceeds to the weighted simulated annealing procedure to improve the current solution π (line 15). Finally, EWSA updates π^* and determines whether the current π_{elite} needs to be reset by an elite solution preservation procedure ElitePreserve() (line 19).

Algorithm 1 The main framework of the EWSA algorithm

Input: A CPP instance
Output: The best solution π^* found so far
1: $\pi \leftarrow$Generate an initial solution
2: $\pi^* \leftarrow \pi$, $\pi_{elite} \leftarrow \pi$, $list_{elite} \leftarrow \varnothing$, $sc \leftarrow 2$
3: $(T_1, T_2) \leftarrow$Based on different target percentages of stable vertices (p_1 and p_2), the corresponding initial temperatures are obtained through a bisection method
4: $T_{final} \leftarrow$The temperature at which the best objective was last updated in WSA($K(\pi^*), \pi, \pi_{elite}, T_2, T_{lb}, sc$)
5: $unImprove \leftarrow 0$, $\pi_{worse} \leftarrow \pi_{elite}$
6: **while** elapsed time does not exceed $time_{limit}$ **do**
7: **if** rand$(0,1) > rate_{c2}$ **then**
8: $T_{init} \leftarrow T_1$, $sc \leftarrow 1$ /* $config_1$ */
9: **else**
10: $unImprove \leftarrow unImprove + 1$
11: $\pi \leftarrow \pi_{elite}$, $T_{init} \leftarrow T_2$, $sc \leftarrow 2$ /* $config_2$ */
12: **end if**
13: $K_{pr} \leftarrow K(\pi^*) + 1$
14: $\pi \leftarrow$ Ensure all vertices in π are within K_{pr} partitions
15: $\pi, \pi_{elite} \leftarrow$ WSA($K_{pr}, \pi, \pi_{elite}, T_{init}, T_{final}, sc$)
16: **if** $f(\pi_{worse}) > f(\pi_{elite})$ **then**
17: $\pi_{worse} \leftarrow \pi_{elite}$
18: **end if**
19: ElitePreserve($\pi_{elite}, \pi^*, unImprove, list_{elite}$)
20: **end while**
21: **return** π^*

Initialization

Since $K(\pi^*)$ is equal to $K(\pi)$ at the beginning of EWSA, the construction of the initial solution π is important. Starting from an empty partition, at each iteration our initial solution generator randomly selects a vertex and adds it to either a new partition or an existing one based on the principle of maximizing the objective value in Eq. (3) (Algorithm 1, line 1). When there are multiple equal solutions, we prefer to add the vertex to a new partition. The construction is completed after all vertices are added to the solution.

The initial temperature is a key parameter for simulated annealing. We propose a method for calculating the initial temperature based on the idea of stable vertices, which obtains the temperature when a certain percentage of vertices reach a stable state. A stable vertex means that any move of that vertex will lead to a worse objective value. The percentage of stable vertices at different temperatures provides a clear indication of how temperature affects the search process. By controlling this percentage, the initial temperature can be adjusted more precisely to achieve better results. Our algorithm calculates two initial temperatures T_1 and T_2 for two search configurations by different target percentages p_1 and p_2 (Algorithm 1, line 3), which are described as follows.

Similar to MDMCP, we also use an automated binary search to find a suitable T_1 (T_2) value in the range $[1, 2000]$ for a given instance. For this purpose, we run WSA where both initial and final temperatures are set to the middle value in the initial range $[1, 2000]$. If it leads to the percentage of stable vertices in the solution near p_1 (p_2), then T_1 (T_2) is set definitively to this value. Otherwise, we re-run WSA with the first or second half-sized range according to whether the percentage of stable vertices is lower or higher than p_1 (p_2).

Partition Restriction and Neighborhood Structure

Previous algorithms, such as MDMCP and SACC, always introduce an empty set to dynamically optimize the number of partitions during simulated annealing. This tends to distribute vertices into more partitions due to high temperature in the early search period while the number of partitions decreases significantly in the later search period so that the final number of partitions is smaller than that in the early search period.

In order to reduce the search space and increase the search efficiency, we propose an adaptive partition strategy to restrict the maximum number of partitions K_{pr} in the solution. Although the optimal number of partitions is unknown, we try to approximate it by the number of partitions of the historical best solution $K(\pi^*)$. To avoid K_{pr} being overfitted to $K(\pi^*)$, an empty partition is introduced to relax the maximum number of partitions which allows for potential improvements. Therefore, before each round of simulated annealing, the partition restriction strategy dynamically presets the maximum number of partitions K_{pr} as the number of partitions of the historical best solution $K(\pi^*)$ plus an empty partition. During this round of simulated annealing, EWSA restricts all vertices to move within these K_{pr} partitions. This partition restriction strategy reduces the search space and speeds up the search.

Neighborhood structure plays an important role in local search-based metaheuristics. In previous algorithms such as MDMCP and SACC, the best neighborhood move of a random vertex is always performed. We still follow this best improvement strategy in our algorithm. Since the maximum number of partitions K_{pr} is fixed in advance, we can define an auxiliary $K_{pr} \times N$ array W, where $W_{k,v}$ represents the sum of the weights of vertex v with the vertices in C_k:

$$W_{k,v} = \sum_{\substack{u \in C_k \\ v \neq u}} w_{v,u} \tag{4}$$

Therefore, the evaluation of each move only requires comparing the sum of weight $W_{k,v} (1 \leq k \leq K_{pr})$ in K_{pr} groups. When vertex v is moved from C_{old} to C_{new}, $W_{old,i}$ and $W_{new,i}$ need to be updated :

$$W_{old,i} = W_{old,i} - w_{i,v} \ \ (1 \leq i \leq N) \tag{5}$$

$$W_{new,i} = W_{new,i} + w_{i,v} \ \ (1 \leq i \leq N) \tag{6}$$

It is easy to find that the time complexity of updating W is $O(N)$ and the time complexity of each evaluation is approximately $O(K_{pr})$.

Weighted Simulated Annealing

We propose a weighting strategy to improve the scoring function in traditional simulated annealing, where the best neighborhood move of a random vertex is always performed which is the same as previous algorithms. Despite of its good performance, the best improvement strategy tends to drive

the search towards premature convergence, resulting in a local optimum trap. The scoring function in traditional simulated annealing relies on the change in the objective value and the current temperature to calculate the acceptance probability, which decides whether to accept the new solution or not:

$$P = exp(\frac{\Delta}{T}) \tag{7}$$

This scoring function mitigates the impact of the best improvement strategy, but the search still tends to converge prematurely, because the traditional scoring function exhibits uneven performance at high and low temperatures in solving the CPP: when the temperature is high, the algorithm always performs poorly; However, when the temperature drops below a threshold, the search converges rapidly.

To improve the traditional simulated annealing scoring function, we introduce a weighting strategy that incorporates the number of times, for which a vertex has not been moved, as a weight in the scoring function. This approach considers not only the change in the objective value and the current temperature but also the times for which a vertex has not been moved, which guides WSA to explore diverse solutions and thus is able to enhance the diversification of the search. By adding this weight, the convergence of the search process can become smooth, which avoids the drawback of uneven performance at high and low temperatures in traditional simulated annealing. Then, we design a new scoring function with varying levels of weighting for the two configurations, detailed as follows:

$$P = \begin{cases} exp(\frac{\Delta}{T}) + min(\frac{T-T_{final}}{T_{init}-T_{final}}, \frac{weight[v]}{\theta}), & sc = 1 \\ exp(\frac{\Delta}{T \cdot \epsilon^{weight[v]}}), & sc = 2 \end{cases} \tag{8}$$

The scoring function increases the probability of accepting new solutions with different degrees. When sc is set to 1, the weight of the vertex is normalized with its expected evaluation count at the current temperature to generate an additional acceptance probability, which is capped by a temperature-dependent threshold to ensure overall convergence. This significantly enhances the randomness and diversification of WSA. When sc is set to 2, the weight influences the acceptance probability through slight temperature adjustments, which enhances intensification of the search compared to the scenario with $sc = 1$. By setting sc to different values for $config_1$ and $config_2$, respectively, EWSA is able to strike a balance between diversification and intensification of the simulated annealing search process.

The framework of WSA is presented in Algorithm 2. Similar to traditional simulated annealing, WSA has two layers, T_{final} and I_{in} are the cutoff parameters of the outer and inner loops (lines 4,6), respectively. In WSA, vertex v is randomly selected from V (line 7). Besides, the fast neighborhood evaluation is applied on π to find the partition C_{new} with the best objective value gain Δ according to W without performing the move (line 8). Subsequently, the insertion move and updating of the auxiliary array W (lines 11-12) are performed only in the case of a positive gain (i.e., $\Delta \geq 0$) or acceptance of a poor solution (line 10). WSA records the number of iterations for which each vertex is evaluated without being moved as its weight (line 18), and resets the weight to zero after each vertex move (line 16).

Algorithm 2 Weighted Simulated Annealing (WSA)

Input: The number of partitions K_{pr}, solutions π and π_{elite}, initial and final temperatures T_{init} and T_{final}, scoring function indicator sc
Output: The elite solution π_{elite} and the current solution π

1: Initialize W according to π and K_{pr} by Eq. (4).
2: $T \leftarrow T_{init}$
3: $I_{in} \leftarrow N \cdot \theta$ /* Calculating the inner loop cutoff condition */
4: **while** $T \geq T_{final}$ **do**
5: $weight \leftarrow$ Array of size N to be with value zero
6: **for** $j = 0 \rightarrow I_{in}$ **do**
7: $v \leftarrow rand[1, N]$
8: $C_{new}, \Delta \leftarrow$Find the best neighborhood move for vertex v within K_{pr} partitions according to W
9: Calculate P according to sc by Eq. (8)
10: **if** rand$(0, 1) \leq P$ **then**
11: $\pi \leftarrow \pi \oplus$ Move(v, C_{old}, C_{new})
12: Update W by Eqs. (5) and (6)
13: **if** $f(\pi) > f(\pi_{elite})$ **then**
14: $\pi_{elite} \leftarrow \pi$
15: **end if**
16: $weight[v] \leftarrow 0$
17: **else**
18: $weight[v] \leftarrow weight[v] + 1$
19: **end if**
20: **end for**
21: $T \leftarrow T \cdot \lambda$
22: **end while**

Two Configurations and Oscillation Strategy

Since the solution space of the CPP is huge, it is important to strive for a balance between exploration and exploitation of the search. However, a single configuration cannot usually satisfy the above two requirements simultaneously. To solve this problem, we design two configurations $config_1$ and $config_2$, and alternate them by an adaptive rate $rate_{c2}$. Each configuration is designed with a specific objective: $config_1$ aims to enhance the diversification of the search, which extensively explores the solution space to find more potential elite solutions, while $config_2$ provides a strong intensification for the search, which focuses on refining the quality of the current elite solution. Although each configuration has its limitations, their roles are complementary with each other. By combining these two configurations, the algorithm can not only thoroughly explore the solution space but also effectively enhance the quality of the elite solution, achieving a synergistic optimization effect.

The specific descriptions of the two configurations are as follows (Algorithm 1, lines 8 and 11). $config_1$ focuses on discovering better elite solutions by increasing the probability of vertex moving to explore wider search space. To achieve this goal, a lower percentage of stable vertices p_1 is

used to generate the initial temperature T_1. Besides, to further enhance the diversification of the search, sc is set to 1. Since $config_1$ is not sensitive to the initial solution, we use the results from the previous round of search as its initial solution. $config_2$ aims to improve the elite solution, so its initial solution is set to the current elite solution. During the search process, to take advantage of the current vertices partitioning, a higher percentage of stable vertices p_2 is used to generate the initial temperature T_2. During the refinement of elite solutions, it is essential to avoid excessive diversification of the seach. Hence, sc is set to 2.

EWSA alternates between the two configurations by a rate $rate_{c2}$ (Algorithm 1, line 7). An oscillation strategy is adopted to adjust the rate $rate_{c2}$ by the quality of the current elite solution and the running time of EWSA. To evaluate the quality of the current elite solution, we record both the worst and best solutions π_{worst} and π^* among the past elite solutions. The quality coefficient is calculated by assessing the relative improvement of $f(\pi_{elite})$ compared to $f(\pi_{worst})$, and then normalizing this measure with respect to $f(\pi^*)$. In addition, EWSA considers the running time. As the running time $time_{past}$ increases, the solution quality evaluation becomes more accurate. The formula for calculating the rate is as follows:

$$rate_{c2} = \frac{f(\pi_{elite}) - f(\pi_{worst})}{f(\pi^*) - f(\pi_{worst})} \times min(\frac{time_{past}}{time_{limit}} \times 2, 1) \tag{9}$$

When $rate_{c2}$ is high, it indicates that the quality of the current elite solution is good. In this case, EWSA tends to search for potential better solutions near the current elite solution, so EWSA will choose $config_2$. Conversely, when $rate_{c2}$ is low, it indicates that the quality of the current elite solution is poor. It tends to escape from the current elite solution to search for the global optimal solution. In this case, EWSA will choose $config_1$. Through this oscillation strategy, EWSA can achieve a fair balance between search exploration and exploitation.

Elite Solution Preservation

Since $config_2$ focuses on improving the current elite solution, to prevent the search from getting stuck near the current elite solution, we introduce an elite solution preservation mechanism, which sets a threshold for the number of iterations without improvement under $config_2$. If this threshold is exceeded, the elite solution is randomly initialized. To avoid falling into the same local optimal trap again, the preservation mechanism introduces a tabu strategy, which forbids searching around the previous elite solutions.

Algorithm 3 presents the procedure of resetting the elite solution. If π_{elite} is updated in WSA, $unImprove$ will be reset. It should be noted that if the elite solution improves the historical best solution, $unImprove$ will be reset to a smaller value sv (line 6); Otherwise, it will be reset to 0 (line 3). This allows EWSA to conduct more searches near the high-quality elite solution and find better solutions. At the same time, SimilarityCheck() will check the similarity between π_{elite} and each solution in $list_{elite}$, and then return the highest similarity rate $rate_{sim}$ (line 4). When π_{elite} has not been improved for $maxUI$ rounds or the similarity rate $rate_{sim}$ is higher than the upper limit α, π_{elite} will be randomly initialized (lines 14-15). But π_{elite} is only added to elite solution set $list_{elite}$ when $rate_{sim}$ is less than α (lines 11-13). SimilarityCheck() calculates the similarity rate as follows. First, we define the intersection function $I(A, B)$:

$$I(A, B) = \{x | x \in A \cap B\} \tag{10}$$

Then, the formula for calculating the similarity rate between two solutions π_1 and π_2 is given as:

$$rate_{sim} = \frac{1}{2 \cdot n} \cdot (\sum_{A_i \in \pi_1} max\{|I(A_i\ ,\ B)|\ , B \in \pi_2\} + \sum_{A_i \in \pi_2} max\{|I(A_i\ ,\ B)|\ , B \in \pi_1\}) \tag{11}$$

Algorithm 3 Elite Preservation

Input: solutions π_{elite} and π^*, unimproved counter $unImprove$, the historical elite solutions $list_{elite}$
Output: solutions π_{elite} and π^*, unimproved counter $unImprove$, the historical elite solutions $list_{elite}$

1: $rate_{sim} \leftarrow 0$
2: **if** $f(\pi_{elite})$ is updated **then**
3: $\quad unImprove \leftarrow 0$
4: $\quad rate_{sim} \leftarrow$ SimilarityCheck($list_{elite}, \pi_{elite}$) according to Eqs. (10) and (11).
5: $\quad$ **if** $f(\pi_{elite}) > f(\pi^*)$ **then**
6: $\quad\quad unImprove \leftarrow sv$
7: $\quad\quad \pi^* \leftarrow \pi_{elite}$
8: $\quad$ **end if**
9: **end if**
10: **if** $rate_{sim} \geq \alpha$ or $unImprove \geq maxUI$ **then**
11: $\quad$ **if** $rate_{sim} < \alpha$ **then**
12: $\quad\quad list_{elite} \leftarrow list_{elite} \cup \pi_{elite}$
13: $\quad$ **end if**
14: $\quad \pi_{elite} \leftarrow$ Randomly generate a solution with $K(\pi^*)$ partitions
15: $\quad unImprove \leftarrow 0$
16: **end if**

Computational Results

Experimental Protocol and Benchmarks

Our EWSA[1] algorithm is implemented in C++ and runs on an Intel Xeon E5-2697 processor with 2.60 GHz CPU and 2 GB RAM. We divide the instances into two categories. The first category includes 161 instances, primarily derived from real-world applications, including aggregation of binary relations, machine cell formation and cluster editing, respectively. These instances are small scale ones and easy to solve. More details can be found in Sørensen and Letchford (2024). The second category consists of 94 challenging instances with random weights. A brief introduction to these

[1]The source code and detailed experimental results can be found at https://github.com/CBer-cn/CPP

vertices	cutoff time(s)	vertices	cutoff time(s)
$n \leq 300$	200	$1000 < n \leq 1500$	4000
$300 < n \leq 500$	500	$1500 < n \leq 2500$	10000
$500 < n \leq 800$	1000	$2500 < n$	20000
$800 < n \leq 1000$	2000		

Table 1: Cutoff time in seconds for instances with different sizes.

Param	Description	Candidate values	Final
T_{lb}	lower bound of temperature	{0.01, 0.1, 1}	0.01
p_1	percentage of stable vertices	{0.7, 0.75, 0.8}	0.8
p_2	percentage of stable vertices	{0.85, 0.9, 0.95}	0.9
θ	coefficient for the inner loop	{50, 100, 150, 200}	100
ϵ	weighting coefficient for WSA	{1.005, 1.01, 1.015, 1.02}	1.01
λ	cooling ratio of WSA	{0.96, 0.97, 0.98, 0.99}	0.98
α	similarity rate threshold	{0.6, 0.7, 0.8, 0.9}	0.9
$maxUI$	maximum unimproved counter	{5, 10, 15, 20, 25}	15
sv	start counter	{0, -5, -10, -15}	-10

Table 2: Parameters tuning results.

benchmarks can be found in Gao et al. (2022). The cutoff times in seconds of tested algorithms are listed in Table 1.

Parameter Tuning

We use the automatic configuration tool "irace" (López-Ibáñez et al. 2016) to calibrate the parameters of the EWSA algorithm. It is conducted by running irace on 16 randomly selected instances of different scales with a total tuning budget of 3200 runs. We adopt the best parameter configuration given by irace as shown in Table 2.

Comparison with the Best Known Solutions

Comparisons between the best solutions found by EWSA and the best known solutions (BKS) are given in Table 3. For the first category of instances, the BKS can be found in Sørensen and Letchford (2024) and Jovanovic, Sanfilippo, and Voß (2023). In the first category of 161 instances, there are 125 instances whose optimal solutions are known. For these instances, EWSA is able to find all the proven optimal solutions. For the remaining 36 instances, EWSA improves and matches the BKS for 6 and 30 instances, respectively. For the second category of instances, to the best of our knowledge, the BKS in the literature are achieved by MDMCP, FSS, and SACC with its variants. For these instances, EWSA improves and matches the BKS for 26 and 68 instances, respectively. In sum, EWSA improves and matches the previous BKS for 32 and 223 out of 255 instances, respectively, without worse solutions.

Instance	Comparison	Better	Equal	Worse
First (161)	proven optimal	0	**125**	0
	unproven optimal	**6**	30	0
Second (94)	unproven optimal	**26**	68	0

Table 3: Comparison results of EWSA with BKS on all the tested 255 instances.

	EWSA		SACC		FSS		MDMCP	
Instance	Best	Avg	Best	Avg	Best	Avg	Best	Avg
First (161)	**161**	**158**	-	-	158	157	158	**158**
Second (94)	**94**	**77**	67	41	57	38	53	29

Table 4: Summary of comparison among EWSA, SACC, FSS and MDMCP on all the tested instances.

Instance	Algorithm	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
First (161)	FSS	3	0	1.1E-1	4	2	3.5E-1
	MDMCP	3	0	8.3E-2	2	2	7.2E-1
Second (94)	SACC	27	0	5.6E-6	50	15	1.1E-8
	FSS	37	0	1.1E-7	55	10	6.5E-10
	MDMCP	41	0	2.4E-8	63	8	1.6E-11

Table 5: Wilcoxon signed ranks test results of EWSA and the reference algorithms in terms of both the best and the average solutions on 255 instances, with a significance level of 0.05.

Comparison with the State-of-the-Art Algorithms

To the best of our knowledge, there are three recent best-performing CPP algorithms: SACC (Gao et al. 2022), FSS (Jovanovic, Sanfilippo, and Voß 2023), and MDMCP (Lu, Zhou, and Hao 2022). Since the source code of SACC is not available to us, we only apply EWSA, MDMCP and FSS on each instance with 20 independent runs in the same environment. The cutoff time (in seconds) of the tested algorithms is listed in Table 1, which is set as that in previous algorithms, such as SACC and MDMCP. Although the source code of SACC is not available to us, according to the information provided by PassMark (2024), the CPU used in SACC (Gao et al. 2022) is 1.28 times faster than ours. Therefore, it is fair to directly use the results reported in SACC for comparison.

Table 4 records the number of times for which EWSA, SACC, FSS, and MDMCP achieve the best results for the best and average objective values on the tested instances. Table 5 presents the comparison of EWSA with SACC, FSS, and MDMCP in terms of the average and best objective values, respectively. R^+_{best} and R^+_{avg} record the number of times for which EWSA 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 EWSA performs worse than the reference algorithms in terms of the best and average objective values.

For the first category of instances, the results are not reported in SACC, so we only compare the results of EWSA, FSS, and MDMCP. From Table 4, we can find that EWSA achieves the best results in terms of the best objective values on all the 161 instances and EWSA performs comparably with FSS and MDMCP in terms of the average objective values. Table 5 shows that all the p-values are greater than 0.05. The reason might be that this category of instances is easy to solve, and all the algorithms are able to obtain the same objective values.

For the second category of instances, from Table 4, one observes that EWSA significantly outperforms other algo-

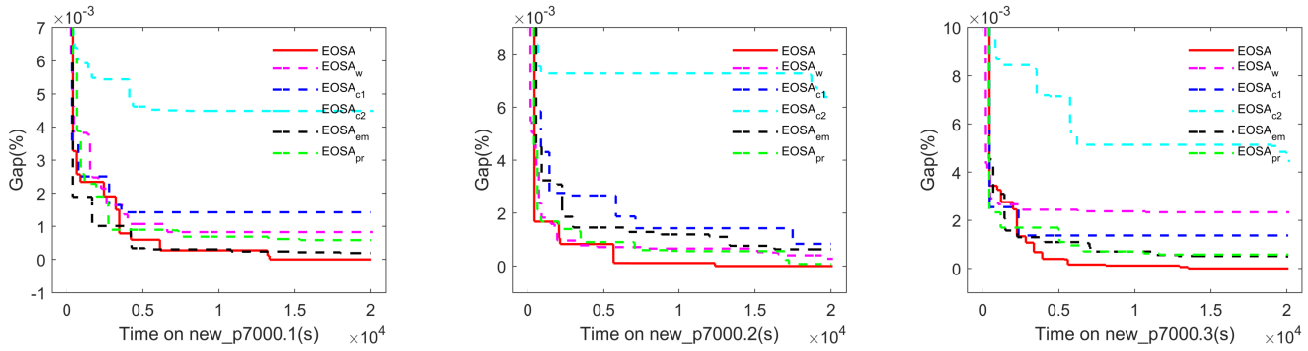

Figure 1: Evolution of objective values by EWSA and its 5 variants on three representative instances.

Comparison	R^+_{best}	R^-_{best}	R^+_{avg}	R^-_{avg}
EWSA$_w$	20	4	25	1
EWSA$_{c1}$	25	0	26	0
EWSA$_{c2}$	26	0	26	0
EWSA$_{em}$	15	5	22	4
EWSA$_{pr}$	15	6	14	12

Table 6: Comparison results of EWSA and other five variants on 26 representative instances.

rithms, since EWSA obtains the best results for the most number of times in terms of both the average and best objective values and its hitting rate is much greater than those of other algorithms. Table 5 shows that all the p-values are much smaller than 0.05, which indicates that there are statistically significant differences among the four algorithms.

Discussion and Analysis

In this section, we present an analysis of each important component in EWSA, including the weighting strategy, the two configurations, the elite solution preservation mechanism, and the partition number restriction strategy. We consider five variants of EWSA, each disabling a different component and keeping other components unchanged.

- **EWSA**$_w$: disable the weighting strategy in WSA.
- **EWSA**$_{c1}$: only using $config_1$, and $rate_{c2}$ is set to 0.
- **EWSA**$_{c2}$: only using $config_2$, and $rate_{c2}$ is set to 1.
- **EWSA**$_{em}$: disable the elite solution preservation mechanism and only keep the historical best solution.
- **EWSA**$_{pr}$: use adaptive partitioning instead of the partition restriction strategy.

We carried out experiments on 26 instances of the second category for which EWSA improves the best known solutions, which are challenging and representative. The running settings of these variants are the same as those of EWSA. Table 6 presents the comparison of each variant with EWSA. R^+_{best} and R^+_{avg} (R^-_{best} and R^-_{avg}) record the number of times for which EWSA obtains better (worse) results than other variants in terms of the best and average objective values.

From Table 6, we can observe that EWSA outperforms other variants, because EWSA obtains better results with the most number of times in terms of both the average and best objective values and its hitting rate is much greater than those of other variants. This indicates that each strategy in EWSA is important. To further compare each variant, Figure 1 depicts the trend of the objective value on three representative instances (new_p7000.1, new_p7000.2, new_p7000.3) when applying EWSA and other variants. Due to the large value of the objective function, each point (x, y) on the curve represents that the gap between the objective value of the current solution and the best solution found by EWSA.

From Figure 1, we can observe that EWSA always achieves the best results when compared with other variants. Specifically, during the convergence process, EWSA consistently outperforms EWSA$_w$, EWSA$_{c1}$, and EWSA$_{c2}$. Besides, EWSA$_{c2}$ always obtains the worst solution. The reason that EWSA outperforms EWSA$_w$ might be that the weighting strategy enhances the diversification of the simulated annealing search process. Additionally, $config_2$ focuses on exploiting the current elite solution, which may lead the search to fall into the local optimum trap, while $config_1$ overly explores diverse solution spaces, which result in overlooking high-quality solutions. Despite that in the first 5000 seconds, EWSA may be slightly worse than EWSA$_{em}$ and EWSA$_{pr}$, EWSA can outperform them after 5000 seconds. The reason might lie in the fact that EWSA$_{em}$ only retains the historical best solution, which speeds up convergence but increases the probability of getting stuck in the local optimum trap. However, the elite solution preservation mechanism enables the search to escape from the local optimum trap. Compared to EWSA$_{pr}$, the partition restriction strategy of EWSA significantly reduces the search space, which improves the search efficiency.

Conclusions

We have proposed an elite-guided weighted simulated annealing algorithm called EWSA for solving the CPP. By combining two configurations, our EWSA helps the search to reach a trade-off between exploitation and exploration. In addition, the diversification of EWSA is enhanced by the weighting strategy. Finally, the partition restriction strategy and the elite solution preservation mechanism further enhance the effectiveness of the algorithm. Computational experiments show the high performance of EWSA 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 National Natural Science Foundation of China (NSFC) under Grant 6240072662, 72101094 and 62202192, the Special Project for Knowledge Innovation of Hubei Province under Grant 2022013301015175, and Interdiciplinary Research Program of Hust 5003300129.

References

Alduaiji, N.; Datta, A.; and Li, J. 2018. Influence propagation model for clique-based community detection in social networks. *IEEE Transactions on Computation Social Systems*, 5(2): 563–575.

Amorim, S.; Barthélemy, J.-P.; and Ribeiro, C. 1992. Clustering and clique partitioning: simulated annealing and tabu search approaches. *Journal of Classification*, 9: 17–41.

Brusco, M.; and Koehn, H. 2009. Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem. *Psychometrika*, 74: 685–703.

Charon, I.; and Hudry, O. 2001. The noising methods: a generalization of some metaheuristics. *European Journal Of Operational Research*, 135(1): 86–101.

Charon, I.; and Hudry, O. 2006. Noising methods for a clique partitioning problem. *Discrete Applied Mathematics*, 154(5): 754–769.

Dorndorf, U.; Jaehn, F.; and Pesch, E. 2008. Modelling robust flight-gate scheduling as a clique partitioning problem. *IEEE Transactions on Plasma Science*, 42(3): 292–301.

Dorndorf, U.; and Pesch, E. 1994. Fast clustering algorithms. *INFORMS Journal on Computing*, 6(2): 141–153.

Fox, J.; Roughgarden, T.; Seshadhri, C.; Wei, F.; and Wein, N. 2020. Finding cliques in social networks: a new distribution-free model. *SIAM Journal on Scientific Computing*, 49(2): 448–464.

Gao, J.; Lv, Y.; Liu, M.; Cai, S.; and Ma, F. 2022. Improving simulated annealing for clique partitioning problems. *Journal of Artificial Intelligence Research*, 74: 1485–1513.

Glover, F. 1989. Tabu search - part I. *INFORMS Journal on Computing*, 1(3): 190–206.

Glover, F. 1990. Tabu search - part II. *INFORMS Journal on Computing*, 2(1): 4–32.

Grötschel, M.; and Wakabayashi, Y. 1989. A cutting plane algorithm for a clustering problem. *Mathematical Programming, Series B*, 45(1-3): 59–96.

Grötschel, M.; and Wakabayashi, Y. 1990. Facets of the clique partitioning polytope. *Math. Program*, 47: 367–387.

Hu, S.; Wu, X.; Liu, H.; Li, R.; and Yin, M. 2021. A novel two-model local search algorithm with a self-adaptive parameter for clique partitioning problem. *Neural Computing & Applications*, 33(10): 4929–4944.

Ion, A.; Carreira, J.; and Sminchisescu, C. 2011. Image segmentation by figure-ground composition into maximal cliques. In Metaxas, D. N.; Quan, L.; Sanfeliu, A.; and Gool, L. V., eds., *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, 2110–2117.

Jaehn, F.; and Pesch, E. 2013. New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics*, 161(13-14): 2025–2037.

Ji, X.; and Mitchell, J. E. 2007. Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Discrete Optimization*, 4(1): 87–102.

Jovanovic, R.; Sanfilippo, A.; and Voß, S. 2023. Fixed set search applied to the clique partitioning problem. *European Journal Of Operational Research*, 309(1): 65–81.

Kirkpatrick, S.; Jr., D. G.; and Vecchi, M. P. 1983. Optimization by simmulated annealing. *Science*, 220(4598): 671–680.

Lu, Z.; Zhou, Y.; and Hao, J. 2022. A hybrid evolutionary algorithm for the clique partitioning problem. *IEEE Transactions on Cybernetics*, 52(9): 9391–9403.

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.

Marcotorchino, J. F.; and Michaud, P. 1981. Heuristic approach of the similarity aggregation problem. *Methods of Operations Research*, 43: 395–404.

Mehrotra, A.; and Trick, M. A. 1998. Cliques and clustering: a combinatorial approach. *Operations Research Letters*, 22(1): 1–12.

Miyauchi, A.; Sonobe, T.; and Sukegawa, N. 2018. Exact Clustering via Integer Programming and Maximum Satisfiability. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32: 1387–1394.

Ouyang, G.; Dey, D. K.; and Zhang, P. 2020. Clique-based method for social network clustering. *Journal Of Classification*, 37(1): 254–274.

Palubeckis, G.; Ostreika, A.; and Tomkevičius, A. 2014. An iterated tabu search approach for the clique partitioning Problem. *The Scientific World Journal*, 2014: 353101.

PassMark. 2024. CPU benchmark. https://www.cpubenchmark.net/, last accessed on 2024-8-1.

Regnier, S. 1983. Sur quelques aspects mathématiques des problèmes de classification automatique. I, II. *Mathématiques et Sciences Humaines*, 82: 31–44.

Skiscim, C. C.; and Golden, B. L. 1983. Optimization by simulated annealing: a preliminary computational study for the TSP. In Roberts, S. D.; Banks, J.; and Schmeiser, B. W., eds., *Proceedings of the 15th Conference on Winter simulation, WSC 1983, Arlington, VA, USA, December 12-14, 1983*, 523–535.

Sørensen, M. M.; and Letchford, A. N. 2024. CP-Lib: Benchmark Instances of the Clique Partitioning Problem. *Mathematical Programming Computation*, 16(1): 93–111.

Strickland, D. M.; Barnes, E. R.; and Sokol, J. S. 2005. Optimal protein structure alignment using maximum cliques. *Operational Research*, 53(3): 389–402.

Sukegawa, N.; Yamamoto, Y.; and Zhang, L. 2013. Lagrangian relaxation and pegging test for the clique partitioning problem. *Advances In Data Analysis And Classification*, 7(4): 363–391.

Wang, H.; Alidaee, B.; Glover, F.; and Kochenberger, G. 2006. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18: 77–97.

Zhou, Y.; Hao, J.; and Goëffon, A. 2016. A three-phased local search approach for the clique partitioning problem. *Journal Of Combinatorial Optimization*, 32(2): 469–491.