Front. Comput. Sci., 2025, 19(6): 196403
https://doi.org/10.1007/s11704-024-40238-8

RESEARCH ARTICLE

Improving local search algorithms for clique relaxation problems via group driven initialization

Rui SUN[1,2], **Yiyuan WANG** (✉)[1,2], **Minghao YIN** (✉)[1,2]

1 School of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China
2 Key Laboratory of Applied Statistics of Ministry of Education, Northeast Normal University, Changchun 130117, China

Abstract Clique relaxation problems are important extension versions of the maximum clique problem with extensive real-world applications. Although lots of studies focus on designing local search algorithms for solving these problems, almost all state-of-the-art local search algorithms adopt a common general initialization method. This paper develops a general group driven initialization method for clique relaxation problems. The proposed method uses two kinds of ways to divide vertices into some subgroups by using the useful information of the search procedure and the structure information of a given instance and then constructs a good initial solution by considering the generated group information. We apply the proposed initialization method to two clique relaxation problems. Experimental results demonstrate that the proposed initialization method clearly improves the performance of state-of-the-art algorithms for the clique relaxation problems.

Keywords combinatorial optimization, clique relaxation problems, group driven initialization, local search, group information

1 Introduction

Given a graph $G = (V, E)$, a clique C is a subset of V such that every two distinct vertices in C are adjacent. As a basic concept of graph theory, one of the major concerns for the model of clique is the identification of cohesive subgroup, which has already lots of valuable applications in many fields, such as biological computing [1], chemoinformatics [2], and social network analysis [3,4]. In practice, many real-world applications cannot be directly modeled as the clique because the model of clique is too strict. To overcome the impracticalities stemming from the ideal model of the clique, a series of clique relaxation models based on different kinds of relaxation constraints are proposed to deal with real-world applications such as k-club [5], k-plex [6], k-quasi-clique [7], and k-defective [8].

Since the above clique relaxation problems involve relaxing certain constraints, each of them has distinct real-world applications. For example, the k-plex is a degree relaxation model of clique, and it aims to define a certain degree of local non-connectivity within a subgraph. Thus, it has been applied to the criminal network analysis field, whose major challenge lies in dealing with intelligence that often contains errors or incomplete information [9]. The k-club model restricts the shortest distance between vertices in the subgraph, which reflects the "spread" in real-world scenarios. As a result, the k-club model has been utilized in modeling epidemic control [10] and internet research [11]. The k-quasi-clique and k-defective models have similar types of constraints. The k-quasi-clique model directly employs a density constraint, while the k-defective-clique model can be regarded as an adapted version of the density constraint. Both of these models can be applied to protein detection [12,13].

As is known, the above mentioned clique relaxation problems are NP-hard problems [14–18], which means that, unless P = NP, there is no polynomial-time algorithm for these problems. Recently, state-of-the-art exact algorithms for k-plex [19], k-defective [8], k-quasi-clique [20], and k-club [21] have been proposed. Because of the NP-hardness of these problems, many researchers turn to designing heuristic algorithms for solving the clique relaxation problems to obtain a good solution within an acceptable time limit. During the past decades, many representative heuristic algorithms have been designed for handling different clique relaxation problems, including k-club [5,22–24], k-plex [25–28], and k-quasi-clique [7,29–31]. Among the heuristic algorithms for the maximum k-quasi-clique problem, NuQClq [7] has demonstrated state-of-the-art performance. It utilizes a novel variant of the configuration strategy called BoundedCC and employs a cumulative saturation-based scoring function. As for the maximum k-plex problem, DCCplex [26] and KLS [27] have shown to be the top-performing algorithms. Furthermore, for the k-club problem, NukCP [24] clearly outperforms other heuristic algorithms. It incorporates a dynamic reduction method and a stratified threshold configuration checking strategy.

The general local search framework for clique relaxation problems usually consists of initialization and search procedures. It works as follows: iteratively generate an initial

Received March 8, 2024; accepted June 2, 2024

E-mail: yiyuanwangjlu@126.com; ymh@nenu.edu.cn

solution and then explore a good candidate solution until the termination condition is reached. Nevertheless, the improvements of previous local search algorithms for clique relaxation problems mainly concentrate on designing some strategies for the search procedure, such as cycling strategy [24,26], scoring function [7], reduction strategy [24], and perturbation strategy [25,27]. Compared to lots of novel strategies in the search procedure, most of these algorithms adopt a frequency-based initialization method. Although a good initial solution has a great impact on the results of the subsequent search procedure, there are few works on studying a specialized initialization process for clique relaxation problems.

In this study, to improve the performance of the initialization procedure of local search algorithms for clique relaxation problems, we propose a group driven initialization method that consists of a group-based initialization method $Init_{gdi}$, as well as two group generation and maintenance procedures GP_{sea} and GP_{str}. Both GP_{sea} and GP_{str} consider the useful information of the search procedure and the structure information of a given instance to a certain extent. In detail, vertices are divided into subgroups based on a novel local optimal matrix and the definition of modularity. Based on the group information of vertices, an initial solution is obtained based on a roulette wheel method. We remark that the proposed group driven initialization method does not impact the search procedure, and thus any local search algorithm based on the general local search framework for clique relaxation problems can directly apply this proposed method. In our work, we apply the proposed initialization method to three local search algorithms that achieve the state-of-the-art performance for k-quasi-clique and k-plex, respectively. Extensive experiments are carried out to evaluate the improved algorithms using our proposed initialization method on the benchmarks used in the literature. Results show the superiority of our proposed initialization method over a common general initialization method.

This paper is organized as follows. In Section 2, we introduce some preliminary definitions. In Section 3, we present our group driven initialization method. Finally, in Section 4, we present the experimental results.

2 Preliminary

An undirected graph $G=(V,E)$ consists of a vertex set $V=\{v_1,v_2,\ldots,v_n\}$ and an edge set $E=\{e_1,e_2,\ldots,e_m\}$. The density of graph G is denoted as $dens(G)=\frac{2\times|E|}{|V|\times(|V|-1)}$. For an edge $e=\{v_i,v_j\}$, vertices v_i and v_j are the endpoints of the edge. Two vertices are neighbors if and only if they belong to the same edge. Given a vertex v, its neighborhood is $N_G(v)=\{u\in V\mid\{v,u\}\in E\}$ and its close neighborhood is $N_G[v]=N_G(v)\cup\{v\}$. The degree of a vertex v is defined as the number of its neighbors, denoted as $deg_G(v)=|N_G(v)|$. Given a vertex set $S\subseteq V$, its neighborhood is $N_G(S)=\bigcup_{v\in S}N_G(v)\setminus S$ and its close neighborhood is $N_G[S]=\bigcup_{v\in S}N_G[v]$. Given a vertex set $S\subseteq V$, the induced subgraph $G[S]=(V_S,E_S)$ is a subgraph of G if $V_S=S$ and the edge set E_S includes all the edges in E that have both endpoints in S. Given a vertex set $S\subseteq V$, a group set is defined as $V_{gro}=\{g_1,g_2,\ldots,g_d\}$ where each vertex in S is partitioned into a specific subgroup (e.g., $g_i\in V_{gro}$).

Given a graph $G=(V,E)$ and a fixed constant $k\in(0,1]$, a k-quasi-clique is a subset $S\subseteq V$ such that $dens(G[S])\geqslant k$. The maximum quasi-clique problem (MQCP) aims to find a k-quasi-clique with the most vertices of a given graph.

Given a graph $G=(V,E)$ and a positive integer k, a k-plex is a subset $S\subseteq V$ such that the degree of each vertex in $G[S]$ is at least $|S|-k$. The maximum k-plex problem (MKPP) aims to find a k-plex with the most vertices of a given graph.

2.1 A general local search framework for clique relaxation problems

A general local search framework for clique relaxation problems is shown in Algorithm 1, including two procedures: the initialization procedure (Line 2) and the search procedure (Line 3). In the beginning, an initial feasible solution D_{ini} is generated by calling the initialization method, and then the algorithm uses the search procedure to improve the solution D until some stop criterion is reached. A local best solution obtained by the search procedure is stored in D_{lbest}. At the end of this search trajectory, if the local best solution D_{lbest} is better than the global best solution D^*, D^* is updated by D_{lbest} (Line 4). Finally, the algorithm returns D^* (Line 5).

In the following, we will introduce a general initialization method as below.

A frequency-based initialization method $Init_{freq}(G)$ [7,25–27]. Each vertex $v\in V$ has a property: frequency, denoted as $freq[v]$. It works as follows: 1) at the beginning, $freq[v]=0$, for each $v\in V$; 2) whenever vertex v is moved (i.e., to be added or removed), the value of its $freq$ will be increased by 1. The $Init_{freq}(G)$ method iteratively picks a vertex with the smallest $freq$ value, with ties broken randomly, until an initial solution cannot be extended. In each iteration, the selected vertex needs to satisfy the constraint of clique relaxation problems.

2.2 The roulette wheel selection method

In our work, we mainly use a roulette wheel selection method to decide which vertex is a candidate vertex [32]. The roulette wheel selection method is a stochastic selection method, where the probability for the selection of an individual is proportional to its fitness. Let us consider N individuals, each characterized by its fitness $w_i>0$ where $i\in[1,N]$. The selection probability of the ith individual is defined as $p_i=\frac{w_i}{\sum_{i=1}^{N}w_i}$.

3 A group driven initialization method

In this section, we design a group driven initialization method

Algorithm 1 A general local search framework for clique relaxation problems

Input: Graph $G=(V,E)$, the *cutoff* time
Output: The best found solution D^*
1 **while** elapsed time < *cutoff* **do**
2 $D_{ini}:=Init(G)$;
3 $D_{lbest}:=Search(D_{ini})$;
4 **if** $|D_{lbest}|>|D^*|$ **then** $D^*:=D_{lbest}$;
5 **return** D^*;

for clique relaxation problems, which includes a novel initialization procedure and two group generation and maintenance procedures. Various types of graphs have been employed to evaluate the performance of our proposed algorithms in our experimental section. We observe that most sparse graphs are with special structural features and thus we can easily take full advantage of the structural information to group all vertices and then construct a high-quality initial solution, which can be a good starting point for the subsequent search procedure. However, for the remaining graphs, especially for the dense graphs, we hardly learn some useful structural information from them and thus we turn to refer to the search information between vertices during the search procedure. After that, we mainly group all vertices based on the search information and then generate a good initial solution.

Based on the above considerations, Section 3.1 proposes a group-based initialization procedure Init_{gdi} based on the group information. In Sections 3.2 and 3.3, to obtain the group information, we present two kinds of group generation and maintenance procedures, i.e., search-based group procedure GP_{sea} and structure-based group procedure GP_{str}.

3.1 A novel group-based initialization procedure

For the proposed group-based initialization procedure $Init_{gdi}$, we apply two group procedures, including GP_{sea} and GP_{str} to generate and maintain the group information denoted as $V_{gro} = \{g_1, g_2, \ldots, g_q\}$, where an input parameter q denotes the maximum number of subgroups. In our work, we decide whether to use GP_{sea} or GP_{str} based on the density value of a given graph, which will be further introduced in our experimental section.

At first, assuming that the initial solution in the $Init_{gdi}$ is denoted as D_{ini}. We give a selection rule for the following $Init_{gdi}$.

Roulette wheel selection rule Given a group set $GoodGroup \subseteq V_{gro}$ that will be defined in Line 6 of Algorithm 2, we choose a subgroup g_t from $GoodGroup$ based on the roulette wheel selection where the fitness value of each subgroup $g' \in GoodGroup$ is defined as $|D_{ini} \cap g'| + 1$. Then, for the search-based method GP_{sea}, a vertex v is randomly selected from a given subgroup g_t, whereas for the structure-based method GP_{str}, the algorithm selects a vertex v with the biggest $score_{coh}(v, D_{ini})$ value that will be mentioned in Section 3.2.

Based on a given group set V_{gro} and a selection rule, we present a group-based initialization procedure $Init_{gdi}$ in Algorithm 2. We utilize $CandSet$ to denote the set of all candidate vertices which satisfies the constraint of clique relaxation problems and can be added into D_{ini}. In the initialization procedure, if vertices in a given graph have been already grouped and with probability 1-p, the algorithm calls a frequency-based initialization method (Line 11). Otherwise, the algorithm chooses a random subgroup g_t from V_{gro} and then the first added vertex v_1 is randomly selected from g_t (Line 3). In the loop of adding vertices, if there exist some vertices in $CandSet$ that belong to some subgroups in V_{gro}, then the algorithm chooses a candidate vertex v_2 according to

Algorithm 2 Init_{gdi}

Input: Graph $G = (V, E)$ and group set V_{gro}
Output: An initial solution D_{ini}

```
1  CandSet := V and D_ini := ∅;
2  if V_gro ≠ ∅ ∧ with probability p then
3      select a random vertex v_1 from a random subgroup
         g_t ∈ V_gro;
4      D_ini := D_ini ∪ {v_1} and update CandSet accordingly;
5      while CandSet ≠ ∅ do
6          GoodGroup := {g_i ∈ V_gro | CandSet ∩ g_i ≠ ∅};
7          if GoodGroup ≠ ∅ then
8              select a vertex v_2 based on Roulette Wheel
                 Selection Rule;
9          else select a random vertex v_2 from CandSet ;
10         D_ini := D_ini ∪ {v_2} and update CandSet accordingly;
11 else D_ini := Init_freq(G) ;
12 return D_ini;
```

the **roulette wheel selection rule** (Lines 6–8). Otherwise, a random vertex v_2 is picked (Line 9). Afterward, the selected vertex is added into D_{ini} and the algorithm updates $CandSet$ accordingly (Line 10). If $CandSet$ becomes an empty set, this loop will be terminated. Finally, the algorithm returns an initial solution D_{ini} (Line 12).

3.2 A search-based group procedure GP_{sea}

In some dense graphs, every vertex is adjacent to most vertices, so it is hard to exploit the structure information to divide vertices into some subgroups. Thus, in this subsection, the search information is mainly exploited. We first introduce the intuition of GP_{sea} and the definition and updating rules of a local optimal matrix that can reflect the search information to a certain extent. Then, the group generation and maintenance procedures of GP_{sea} are displayed.

3.2.1 The intuition of GP_{sea}

The intuition behind GP_{sea} is inspired by the backbone structure observed in combinatorial optimization problems [33], which indicates that high-quality solutions often share common components. According to this observation, we consider a scenario where two vertices frequently appear together in some local best solutions. Interestingly, we also find that these pairs of vertices tend to appear in different high-quality local best solutions. As a result, our approach aims to classify these vertices into different subgroups, where each subgroup is associated with different high-quality solutions to some extent. By utilizing GP_{sea}, the objective of $Init_{gdi}$ is to expedite the search process by initializing the initial solution from promising subgroups (i.e., search regions).

3.2.2 A local optimal matrix

In the general search framework for clique relaxation problems, at the end of each search procedure, a local best solution D_{lbest} is obtained. It indicates which vertices are included in the local best solution during the current search trajectory. If two vertices often appear in the local best solution, it not only indicates the importance of these two vertices but also implies that they may have high similarity. According to this search information, we first define a local optimal matrix M of size $n \times n$ where $n = |V|$. An element m_{ij}

corresponds to the occurrence frequency of a pair of vertices v_i and v_j appearing in a local best solution D_{lbest}. It works as follows.

Matrix Rule 1 At the beginning, all the elements in the local optimal matrix are set to 0.

Matrix Rule 2 When D_{lbest} is obtained, for each pair of vertices v_i and v_j where v_i, v_j are two distinct vertices in D_{lbest}, m_{ij} and m_{ji} are increased by 1.

To make readers clearly understand our local optimal matrix, we present an example in Fig. 1.

Given a graph $G = (V, E)$, we use the information of matrix M to divide vertices into at most q subgroups, i.e., $V_{gro} = \{g_1, g_2, \ldots, g_q\}$ where the size of each subgroup is $|V|/q$. During the search procedure, we think the two adjacent vertices often appearing in the same local best solution should be partitioned into the same subgroup. Thus, to greatly group all vertices in a given graph, we give an objective function to measure the quality of the grouping results.

$$obj_{sea}(V_{gro}) = \sum_{t=1}^{q} \left(\sum_{v_i, v_j \in g_t \wedge \{v_i, v_j\} \in E} m_{ij} \right).$$

For a group g_t, its weight is defined as the sum of m_{v_1,v_2} for all vertex pairs (v_1, v_2) connected by an edge within the group. The objective function obj_{sea} is designed to maximize the total weight of all groups.

3.2.3 A Group Generation Procedure for GP_{sea}

To make a good balance between the accuracy and complexity of calculating $obj_{sea}(V_{gro})$, we adopt a greedy group generation procedure to divide vertices into some subgroups. In this procedure, we use a two-level scoring function that combines two kinds of information (i.e., search information and structure information). The primary scoring function considers the search information, whereas the secondary scoring function refers to the structure information.

The primary scoring function based on the local optimal matrix is defined as follows.

$$score_m(v_i, g_t) = \sum_{v_j \in N_G(v_i) \cap g_t} m_{ij}.$$

To maximize the value of objective function $obj_{sea}(V_{gro})$, we apply $score_m$ to decide which vertex v_i is selected and then added into a subgroup g_t. To address the issue of tie-breaking in the primary scoring function, we employ the secondary scoring function to further select a vertex among these vertices with the same best $score_m$. The proposed secondary scoring function considers the cohesive relationship

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	1	2
V_2	1	0	3	0	0
V_3	1	3	0	2	1
V_4	1	0	2	0	2
V_5	2	0	1	2	0

$\xrightarrow{\{V_1, V_4, V_5\}}$

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	2	3
V_2	1	0	3	0	0
V_3	1	3	0	2	1
V_4	2	0	2	0	3
V_5	3	0	1	3	0

Fig. 1 An example for local optimal matrix M. Updating M by using $D_{lbest} = \{v_1, v_4, v_5\}$. The red numbers are the updated elements in M.

of a vertex v_i and a subgroup g_t, which is defined as follows.

$$score_{coh}(v_i, g_t) = |N_G(v_i) \cap g_t|.$$

It is easy to observe that the more vertices in g_t are adjacent to v_i, the bigger $score_{coh}$ value. Based on the above scoring functions, we present a selection rule for the adding process of the proposed greedy group generation procedure.

Selection Rule select a vertex v_i from a candidate set with the biggest $score_m(v_i, g_t)$ value, breaking ties by preferring the one with the highest $score_{coh}(v_i, g_t)$ value, further ties are broken randomly.

The proposed group generation procedure is outlined in Algorithm 3. V_{rem} stores vertices that have not been grouped, which is initialized to V. The group set V_{gro} is initialized to an empty set (Line 1). There is an outer loop (Lines 2–9) and an inner loop (Lines 6−8), where the outer loop aims to generate a series of subgroups and the inner loop is responsible for formulating a certain subgroup. At the beginning of each outer loop, a vertex set g_t is initialized to an empty set. The algorithm selects a vertex $v_i \in V_{rem}$ with the biggest $freq_{best}(v_i)$ value, where the number of times of vertex v_i appearing in all obtained local best solutions is expressed by $freq_{best}(v_i)$ (Line 4). v_i will be added into g_t and then be also deleted from V_{rem} (Line 5). In each inner loop, the algorithm iteratively adds a vertex into g_t, until $|g_t| = |V|/q$ or $N_G(g_t) \cap V_{rem} = \emptyset$ (Lines 6–8). During the vertex selection process, the algorithm employs the BMS strategy [34], i.e., randomly sampling t_1 vertices to compose a candidate set and then selecting a vertex from the candidate set according to the selection rule. After each inner loop, if the size of g_t equals $|V|/q$, then g_t will be stored into V_{gro} (Line 9). At last, the algorithm returns V_{gro} (Line 10).

3.2.4 A novel group-based search framework with $Init_{gdi}$ and GP_{sea}

In this subsection, we will modify the general local search framework by using our proposed group-based initialization procedure and search-based group procedure, resulting in a novel search framework (Algorithm 4). In addition, Algorithm 4 will show how to maintain the group information for GP_{sea} in the general local search framework. We remark that Algorithm 4 does not modify the search procedure of Algorithm 1 (i.e., the boxed parts of Algorithm 4 can be skipped). It means that if a local search algorithm for clique

Algorithm 3 GreedyGroup$_{sea}$

Input: Graph $G = (V, E)$ and local optimal matrix M
Output: Group set V_{gro}
1 $V_{rem} := V$ and $V_{gro} := \emptyset$;
2 **while** $|V_{rem}| \geq |V|/q$ **do**
3 $g_t := \emptyset$;
4 select a vertex $v_i \in V_{rem}$ with the biggest $freq_{best}(v_i)$ value, breaking ties randomly;
5 $g_t := g_t \cup \{v_i\}$ and $V_{rem} := V_{rem} \setminus \{v_i\}$;
6 **while** $|g_t| < |V|/q \wedge N_G(g_t) \cap V_{rem} \neq \emptyset$ **do**
7 select a vertex v_j among t_1 samples from $N_G(g_t) \cap V_{rem}$ based on **Selection Rule**;
8 $g_t := g_t \cup \{v_j\}$ and $V_{rem} := V_{rem} \setminus \{v_j\}$;
9 **if** $|g_t| == |V|/q$ **then** $V_{gro} := V_{gro} \cup g_t$;
10 **return** V_{gro};

relaxation problems is based on the general local search framework, then our proposed group-based initialization procedure and the corresponding grouping generation and maintenance procedures can directly replace the original initialization method of this framework.

Algorithm 4 uses $freq_{inc}$ to record the sum of increments of $freq$ values of all vertices since the last time the greedy group generation procedure $GreedyGroup_{sea}$ was called. To implement it, we define an auxiliary variable $freq_{old}$. At first, the matrix M is initialized according to the matrix rule 1 (Line 1), as well as V_{gro} and $freq_{old}$ are initialized accordingly (Line 2). Whenever a local optimal solution D_{lbest} is obtained, $freq_{inc}$ and M need to be updated (Lines 6–7). If $freq_{inc}$ reaches a specific threshold $\alpha \times |V|$, where α is a parameter, $GreedyGroup_{sea}$ is called to group vertices and $freq_{old}$ needs to be recalculated (Lines 8–10). The matrix M contains little useful information at the beginning of the search procedure and needs to be constantly updated after each search procedure. Thus, $GreedyGroup_{sea}$ should be called within a certain period.

3.3 A structure-based group procedure GP_{str}

In this section, we present a structure-based group procedure GP_{str} for the graphs with obvious structural features. First, we give the intuition of a greedy group generation procedure for GP_{str}. Moreover, we introduce a group updating procedure based on the definition of modularity. Finally, a group-based search framework with $Init_{gdi}$ and GP_{str} is presented.

3.3.1 The intuition of GP_{str}

Massive sparse graphs often exhibit distinctive community structures, characterized by subgroups of vertices with dense connections internally and relatively fewer connections to vertices outside their respective subgroups. To harness these structural characteristics, we utilize GP_{str}, which involves two key steps: $GreedyGroup_{str}$ and $GroupUpdate_{str}$. In detail, $GreedyGroup_{str}$ aims to identify groups of vertices with strong internal connections, while $GroupUpdate_{str}$ iteratively updates these groups to further differentiate the distinct community structures. During the initialization procedure, our focus lies in selecting a series of vertices within a specific subgroup, with the aim of leveraging the inherent community structure.

Algorithm 4 A group-based search framework with $Init_{gai}$ and GP_{sea}

Input: Graph $G = (V, E)$, the *cutoff* time
Output: The best found solution D^*

1 initialize a local optimal matrix M based on **Matrix Rule 1**;
2 $V_{gro} := \emptyset$ and $freq_{old} := 0$;
3 **while** elapsed time < *cutoff* **do**
4 $D_{ini} := Init_{gdi}(V_{gro}, G)$;
5 $D_{lbest} := Search(D_{ini})$;
6 $freq_{inc} := \sum_{v \in V} freq[v] - freq_{old}$;
7 update M based on **Matrix Rule 2**;
8 **if** $freq_{inc} > \alpha \times |V|$ **then**
9 $V_{gro} := GreedyGroup_{sea}(G, M)$;
10 $freq_{old} := \sum_{v \in V} freq[v]$;
11 **if** $|D_{lbest}| > |D^*|$ **then** $D^* := D_{lbest}$;
12 **return** D^*;

3.3.2 A group generation procedure for GP_{str}

To obtain several initial cohesive subgroups, we design a greedy group generation procedure in Algorithm 5. Variable try represents the consecutive times of obtaining an unsatisfied subgroup whose size is less than $|V|/q$. Initially, V_{rem}, try and V_{gro} are initialized accordingly (Line 1). In the process of generating subgroups, a new subgroup g_t is initialized to a vertex in V_{rem} with the biggest deg_G value, and this selected vertex is deleted from V_{rem} (Lines 3–5). Afterward, the algorithm adds a vertex with the biggest $score_{coh}$ value into g_t, which aims to enhance its cohesive structure (Lines 6–8). After the subgroup generating procedure, if $|g_t|$ equals $|V|/q$, then g_t will be stored in V_{gro} and try is reset to 0 (Lines 9–10). Otherwise, try is increased by 1 (Line 11). At last, if try reaches a predefined threshold max_tries, V_{gro} will be returned (Line 12).

3.3.3 A group updating procedure for GP_{str}

To further optimize an initial group set $V_{gro} = \{g_1, \ldots, g_q\}$, our goal is that the vertices belonging to a subgroup are densely connected, whereas vertices belonging to different subgroups are sparsely connected. The concept of modularity, as defined by Newman [35], is commonly utilized to generate community structures. Modularity values range between 0 and 1, with higher values indicating denser internal connections within the same groups and sparser connections between different groups. In our context, since our focus is solely on the grouped vertices, our objective is to maximize the modularity specifically for these grouped vertices. The objective function is defined as follows:

$$obj_{str}(V_{gro}) = \sum_{t=1}^{q} \left[\frac{|E_{g_t}|}{2|E|} - \left(\frac{\sum_{v \in g_t} deg_G(v)}{2|E|} \right)^2 \right],$$

where an induced subgraph of a subgroup g_t is $G[g_t] = (V_{g_t}, E_{g_t})$.

Note that there maybe exists lots of ungrouped vertices in V, denoted by V_{ungr}. That is, $V = \cup V_{gro} \cup V_{ungr}$. In the group updating procedure, we only care about verities in the group set V_{gro} and aim to maximize its objective value obj_{str}.

In the following, we propose a novel way to measure the

Algorithm 5 $GreedyGroup_{str}$

Input: Graph $G = (V, E)$
Output: Group set V_{gro}

1 $V_{rem} := V$, $try := 0$ and $V_{gro} := \emptyset$;
2 **while** $try < max\_tries$ **do**
3 $g_t := \emptyset$;
4 select a vertex $v_i \in V_{rem}$ with the biggest $deg_G(v_i)$ value;
5 $g_t := g_t \cup \{v_i\}$ and $V_{rem} := V_{rem} \setminus \{v_i\}$;
6 **while** $|g_t| < |V|/q \wedge N_G(g_t) \cap V_{rem} \neq \emptyset$ **do**
7 select a vertex v_j among t_1 from $N_G(g_t) \cap V_{rem}$ with the biggest $score_{coh}(v_j, g_t)$ values, breaking ties randomly;
8 $g_t := g_t \cup \{v_j\}$ and $V_{rem} := V_{rem} \setminus \{v_j\}$;
9 **if** $|g_t| == |V|/q$ **then**
10 $V_{gro} := V_{gro} \cup g_t$ and $try := 0$;
11 **else** $try := try + 1$;
12 **return** V_{gro};

change of modularity during the process of adding or removing a vertex.

Adding process When adding a vertex v into a subgroup g_t, we compute the value of modularity by considering a new induced graph $G' = (V', E')$ where $V' = \cup V_{gro} \cup \{v\}$. In detail, before selecting a vertex $v \in V_{ungr}$ to add into a subgroup g_t in V_{gro}, the set $\{v\}$ is considered as an extra subgroup (i.e., g_{q+1}).

Removing process When deleting a vertex v from a subgroup g_t, the value of modularity needs to be recalculated based on a new induced graph $G'' = (V'', E'')$ where $V'' = V_{gro}$. Specifically, after deleting a vertex v from a subgroup g_t in V_{gro}, the set $\{v\}$ is seen as an extra subgroup (i.e., g_{q+1}).

Based on the above assumption, the scoring function $score_{add}(v, g_t)$ of adding a vertex v is defined as follows [36].

$$\begin{aligned} & score_{add}(v, g_t) \\ &= \left[\frac{|E_{g_t}| + L'_{v,g_t}}{2|E'|} - \left(\frac{\sum_{v' \in g_t \cup \{v\}} deg_{G'}(v')}{2|E'|}\right)^2\right] \\ &\quad - \left[\frac{|E_{g_t}|}{2|E'|} - \left(\frac{\sum_{v' \in g_t} deg_{G'}(v')}{2|E'|}\right)^2 - \left(\frac{deg_{G'}(v)}{2|E'|}\right)^2\right] \\ &= \frac{1}{2|E'|}\left(L'_{v,g_t} - \frac{\sum_{v' \in g_t} deg_{G'}(v') \times deg_{G'}(v)}{|E'|}\right), \end{aligned}$$

where $L'_{v,g_t} = |N_{G'}(v) \cap g_t|$. Similarly, we propose a novel deleting scoring function $score_{del}(v, g_t)$ as below.

$$\begin{aligned} & score_{del}(v, g_t) \\ &= \left[\frac{|E_{g_t}| - L''_{v,g_t}}{2|E''|} - \left(\frac{\sum_{v' \in g_t \setminus \{v\}} deg_{G''}(v')}{2|E''|}\right)^2\right. \\ &\quad \left. - \left(\frac{deg_{G''}(v)}{2|E''|}\right)^2\right] - \left[\frac{|E_{g_t}|}{2|E''|} - \left(\frac{\sum_{v' \in g_t} deg_{G''}(v')}{2|E''|}\right)^2\right] \\ &= \frac{1}{2|E''|}\left(\frac{(\sum_{v' \in g_t} deg_{G''}(v') - deg_{G''}(v)) \times deg_{G''}(v)}{|E''|} - L''_{v,g_t}\right), \end{aligned}$$

where $L''_{v,g_t} = |N_{G''}(v) \cap g_t|$.

Based on the definitions of $score_{add}$ and $score_{del}$, we propose the following vertex selection rules.

Add Rule Select a vertex v and a subgroup g_t from a candidate set with the biggest $score_{add}(v, g_t)$ value, breaking ties by preferring the one with the highest $freq$ value.

Delete Rule Select a vertex v and a subgroup g_t from a candidate set with the biggest $score_{del}(v, g_t)$ value, breaking ties by preferring the one with the lowest $freq$ value.

In the adding operation, when multiple vertex-group pairs have the same maximum $score_{add}$ value, our preference is to select a vertex with the highest frequency. This preference is based on the assumption that vertices with higher frequency are more significant in the search procedure, and thus they should be added to subgroups with higher priority. Conversely, when removing a vertex from its group, we prioritize vertices with lower frequency as they are considered less important. The GroupUpdate$_{str}$ is presented in Algorithm 6 where parameter *max_count* is used to control the number of updating V_{gro}. The algorithm adds some vertices and deletes some other vertices to update V_{gro}. Specifically, in each iteration, one vertex within the groups and one vertex outside the groups are swapped. The selection of these two vertices is done using the **Delete Rule** and **Add Rule**, respectively. GroupUpdate$_{str}$ aims to enhance the community structure within the groups.

Algorithm 6 GroupUpdate$_{str}$

Input: Graph $G = (V, E)$ and group set V_{gro}
Output: A modified group set V_{gro}

```
for i = 0; i < max_count; i++ do
    select a vertex v_i among t_1 samples from V_gro based on Delete Rule;
    delete vertex v_i from V_gro;
    select a vertex v_j and a group g_t among t_1 samples from N_G(∪V_gro) based on Add Rule;
    add vertex v_j into g_t where g_t ∈ V_gro;
return V_gro;
```

3.3.4 A novel group-based search framework with $Init_{gdi}$ and GP$_{str}$

A novel search framework based on our proposed group-based initialization procedure and search-based group procedure GP$_{str}$ is shown in Algorithm 7. We mention again that our proposed methods do not change the search procedure of the general search framework (i.e., the boxed parts) for clique relaxation problems. *period* is used to record the execution number of the search procedure and *max_period* is a parameter that represents the frequency of updating V_{gro}. Before the search procedure, V_{gro} is initialized through GreedyGroup$_{str}$. In each *max_period* rounds, the algorithm calls *GroupUpdate* to update V_{gro} (Lines 6–7).

4 Experimental results

We conducted experiments to evaluate the proposed initialization method on a broad range of classic and sparse benchmarks.

Three state-of-the-art local search algorithms, including NuQClq [7] for the MQCP as well as DCCplex [26] and FD-TS [25] for the MKPP, all adopt the general search framework (i.e., Algorithm 1). Thus, as described in Sections 3.2.3 and 3.3.3, our proposed group-based initialization procedure $Init_{gdi}$ and the corresponding group procedures (i.e., GP$_{sea}$ and GP$_{str}$) can be easily applied to the above algorithms, resulting in three novel algorithms including NuQClq-GDI, DCCplex-GDI, and FD-TS-GDI. For a given instance, if its density

Algorithm 7 A group-based search framework with $Init_{gdi}$ and GP$_{str}$

Input: Graph $G = (V, E)$, the *cutoff* time
Output: The best found solution D^*

```
1  period := 1;
2  V_gro := GreedyGourp_str(G);
3  while elapsed time < cutoff do
4      D_ini := Init_gdi(V_gro, G);
5      D_lbest := Search(D_ini);
6      if period%max_period == 0 then
7          V_gro := GroupUpdate(V_gro);
8      period := period + 1;
9      if |D_lbest| > |D*| then D* := D_lbest ;
10 return D*;
```

value is larger than 0.35, the corresponding algorithm uses GP_{sea}. Otherwise, it applies GP_{str}. The source codes of the three algorithms were provided by the authors. For all competitors, we set the same parameters as described in the corresponding literature. All the algorithms are implemented in C++ and compiled by g++ with -O3 option. All experiments are run on Intel Xeon Gold 6238 CPU @ 2.10 GHz with 512 GB RAM under CentOS 7.9.The parameters of our proposed initialization method are tuned by irace [37]. Specifically, for the GP_{str} method, *max_count*, t_1, *max_period*, *max_tries*, q and p are set to 100, 50, 20, 10, 50, and 60% while for the GP_{sea} method, t_1, p, q, and α are set to 50, 60%, 50, and 850, respectively.

For all instances, all the algorithms are performed 10 independent runs with different random seeds from 1 to 10 and the cutoff time is set to 1000 seconds. For each instance, *max* denotes the best size found, and *avg* denotes the average size obtained over 10 runs. When *max* = *avg*, we do not report *avg*. The bold values in the tables indicate the best solution among all the algorithms. We do not report the results when all the algorithms obtain the same solutions. Note that, for the MQCP and MKPP, the same graph with different k is considered as the different instances.

4.1 Results for the MQCP

We collect all used instances from [7], including 187 classic instances from DIMACS benchmark [38] and BHOSLIB benchmark [39] and 102 sparse instances from Florida Sparse Matrix Collection [40] and Stanford Large Network Dataset Collection. Besides, we consider 65 massive sparse graphs from the Network Data Repository [41] with more than 10^5 vertices and more than 10^6 edges, which has been used to test the performance of clique related problems [42,24]. For 65 massive sparse graphs, k is set to 0.9, 0.8, 0.7, and 0.6. The total number of newly selected instances is 260.

The results for NuQClq-GDI and NuQClq on classic instances and sparse instances are reported in Tables 1 and 2, respectively. For classic instances, NuQClq-GDI obtains a better solution than NuQClq on 7 instances and only fails on one instance C2000.9 with $k = 0.999$. As for instances where the algorithms obtain the same best solutions, compared with NuQClq, NuQClq-GDI obtains 10 better average solutions, except for 5 instances. For two sparse benchmarks, NuQClq-GDI achieves better solutions on 13 instances, whereas NuQClq finds better solutions on 3 instances. In terms of average solution, NuQClq-GDI outperforms NuQClq on 23 instances while it is defeated on only 5 instances.

4.2 Results for the MKPP

For the MKPP, we use the same k values as described in the corresponding literature (i.e., $k = 2,3,4$). We adopt several popular benchmarks from [26], including 363 classic instances from DIMACS benchmark [38] and BHOSLIB benchmark [39] and 108 sparse instances from Network Data Repository [41]. According to our results, most of these sparse instances are so easy that all algorithms obtain the same quality values. Thus, we select 65 massive sparse instances as our additional benchmark, which has already been described in the previous

Table 1 Experimental results of NuQClq and NuQClq-GDI on classic instances

Instances	k	NuQClq		NuQClq-GDI		Instances	k	NuQClq		NuQClq-GDI		Instances	k	NuQClq		NuQClq-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
C4000.5	0.8	**53**	**52.9**	**53**	52.7	frb40-19-5	0.999	**40**	39.7	**40**	**40**	frb56-25-5	0.999	**57**	55.7	**57**	**56**
brock400_1	0.999	**27**	**26**	**27**	25.6	frb50-23-4	0.999	**51**	50.9	**51**	**51**	frb59-26-1	0.999	**59**	58.3	**59**	**58.5**
brock800_2	0.999	**24**	**24**	**24**	23.7	frb53-24-1	0.999	53	52.9	**54**	**53.1**	frb59-26-4	0.999	59	58.1	**60**	**58.4**
brock800_3	0.999	22	22	**25**	**22.3**	frb53-24-2	0.999	**54**	53	**54**	**53.1**	frb59-26-5	0.999	**60**	58.7	**60**	**59**
C1000.9	0.999	**70**	69.9	**70**	**70**	frb53-24-5	0.999	**54**	**52.8**	**54**	52.6	hamming10-4	0.95	**88**	**87.3**	**88**	87.1
C2000.9	0.999	**82**	**79.5**	80	**79.5**	frb56-25-1	0.999	56	55.5	**57**	**55.6**	san400_0.7_2	0.95	**65**	64	**65**	**65**
C4000.5	0.9	30	30	**31**	**30.2**	frb56-25-2	0.999	56	55.3	**57**	**55.5**	san400_0.7_3	0.95	**40**	39.6	**40**	**39.9**
frb100-40	0.999	99	98.6	**101**	**98.7**	frb56-25-4	0.999	**57**	55.4	**57**	**55.7**						

Table 2 Experimental results of NuQClq and NuQClq-GDI on sparse instances

Instances	k	NuQClq		NuQClq-GDI		Instances	k	NuQClq		NuQClq-GDI		Instances	k	NuQClq		NuQClq-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
LiveJournal	0.9	**430**	**427.2**	**430**	424.4	rgg-n_2_24_s0	0.9	**28**	**25.5**	**28**	25.4	web-wikipedia_link	0.9	452	250	**660**	**357.9**
	0.8	**472**	471	**472**	**472**		0.8	31	29.6	**34**	**30.6**		0.8	903	312.4	**945**	**511.4**
	0.6	**628**	619.8	**628**	**628**		0.7	**38**	35.2	**38**	**35.4**		0.7	**1214**	**562.1**	1156	512
orkut	0.9	**124**	**116.9**	115	109.7		0.6	**46**	**42.9**	44	41.8		0.6	597	431.4	**1156**	**755.5**
	0.8	**166**	146.3	**166**	**149.1**	soc-orkut	0.9	107	93.4	**114**	**101.9**	web-wikipedia-growth	0.9	47	**36.7**	**49**	33.2
	0.7	201	194.9	**202**	**201.1**		0.8	**159**	129	**159**	**135.9**		0.8	70	59.2	**75**	**61.9**
web-Google	0.8	**69**	68.9	**69**	**69**		0.7	**194**	170.7	**194**	**171.1**	wikipedia_link_en	0.9	**74**	50.5	**74**	**58.7**
wiki-topcats	0.9	**47**	38.4	**47**	**41.3**		0.6	**244**	225.2	**244**	**228.6**		0.8	60	42.2	**111**	**54.6**
	0.8	**68**	54.5	**68**	**60.3**	soc-sinaweibo	0.9	**112**	105.2	**112**	**106.3**		0.7	**150**	69.7	**150**	**92.8**
	0.7	**102**	83.1	**102**	**84.5**		0.8	**183**	122.2	**183**	**122.5**	soc-ljournal-2008	0.9	**475**	435.7	**475**	**460.8**
	0.6	**171**	**143**	**171**	133.2		0.7	**263**	**217.4**	**263**	188.7		0.8	**542**	514.7	**542**	**542**
dbpedia-link	0.9	**57**	**49.5**	**57**	48.5		0.6	241	201	**345**	**218**		0.7	**655**	638.7	**655**	**655**
	0.8	**87**	67.6	**87**	**76.8**	twitter_mpi	0.8	312	312	**501**	**387.6**		0.6	**793**	775.2	**793**	**776.1**
	0.7	**149**	89.2	**149**	**103.2**	web-baidu-baike	0.9	24	24	**28**	**24.4**	soc-twitter-higgs	0.6	**257**	251	**257**	**255.9**
	0.6	**220**	127.8	**220**	**141.3**		0.8	33	33	**34**	**33.1**						

subsection. In total, 195 (65×3) instances are picked.

Table 3 summarizes the results of classic and sparse benchmarks for the MKPP. #inst denotes the number of instances in each benchmark. #better and #worse denote the number of instances where FD-TS-GDI and DCCplex-GDI achieve better and worse maximal (or average) results, respectively. For classic benchmarks, no matter the best or average solutions, DCCplex-GDI and FD-TS-GDI perform better than the corresponding original algorithm. As for sparse benchmarks, most of them for all the algorithms achieve the same solution. Nevertheless, for the instances where our proposed algorithm and the corresponding competitor perform differently, our proposed algorithm steadily outperforms its competitor.

We provide the detailed results for FD-TS-GDI and FD-TS on classic and sparse instances in Tables 4 and 5, respectively. Additionally, the results of DCCplex-GDI and DCCplex on classic and sparse instances are shown in Tables 6 and 7, respectively. In the case of classic instances, FD-TS-GDI and DCCplex-GDI outperform their counterparts in finding better maximal solutions in 34 and 20 instances, respectively. They are only defeated in 5 and 1 instances, respectively. When considering the average solutions for classic instances, FD-TS-GDI and DCCplex-GDI outperform FD-TS and DCCplex

Table 3 Comparative results on all the MKPP benchmarks

Algorithm	Benchmark	#inst	max		avg	
			#better	#worse	#better	#worse
FD-TS-GDI vs.	classic instances	363	**34**	5	**39**	8
FD-TS	sparse instances	303	**3**	1	**11**	7
DCCplex-GDI vs.	classic instances	363	**20**	1	**78**	3
DCCplex	sparse instances	303	**1**	0	**10**	6

Table 4 Experimental results of FD-TS and FD-TS-GDI on classic instances

Instances	k	FD-TS		FD-TS-GDI		Instances	k	FD-TS		FD-TS-GDI		Instances	k	FD-TS		FD-TS-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
brock400_4	2	**33**	32.2	**33**	**33**	san400_0.7_3	2	**27**	26.7	**27**	**27**	gen400_p0.9_65	3	**101**	100.7	**101**	**101**
C1000.9	2	**82**	81.2	**82**	**82**	san400_0.9_1	2	**102**	101.5	**102**	**102**	keller6	3	**91**	**90.3**	90	89.5
C2000.5	2	**20**	**19.5**	**20**	19.3	brock800_1	3	**30**	**29.7**	**30**	29.5	san400_0.7_2	3	**47**	46.1	**47**	**47**
C2000.9	2	**91**	90.2	**91**	**90.6**	brock800_2	3	**30**	**30**	**30**	29.8	brock800_2	4	**34**	**33.6**	**34**	33.4
C4000.5	2	**21**	**20.2**	**21**	20.1	brock800_3	3	**30**	29.9	**30**	**30**	brock800_3	4	**34**	33.8	**34**	**34**
frb100-40	2	**124**	123.1	**124**	**123.2**	C1000.9	3	**95**	94.5	**95**	**95**	C1000.9	4	108	107.2	**109**	**107.7**
frb50-23-1	2	**67**	65.9	**67**	**66.2**	C2000.5	3	**23**	**22.1**	22	22	C2000.9	4	119	118.1	**120**	**118.5**
frb50-23-2	2	**66**	65.5	**66**	**66**	C2000.9	3	**106**	104.5	105	**104.6**	DSJC1000.5	4	**24**	23.8	**24**	**23.9**
frb50-23-3	2	64	**63.9**	**65**	63.8	frb100-40	3	**148**	**146.8**	147	145.9	frb100-40	4	**168**	**167.9**	**168**	167.1
frb50-23-4	2	**66**	65.3	**66**	**65.4**	frb50-23-1	3	**79**	78	**79**	**78.1**	frb50-23-1	4	91	89.8	**92**	**90.7**
frb50-23-5	2	**66**	65.7	**66**	**66**	frb50-23-2	3	78	77.9	**79**	**78.4**	frb50-23-2	4	**91**	89.7	**91**	**90**
frb53-24-1	2	70	69.2	**71**	**70.4**	frb50-23-3	3	**76**	75	**76**	**75.3**	frb50-23-3	4	**87**	**86.3**	**87**	86.2
frb53-24-2	2	**70**	68.7	**70**	**69.2**	frb50-23-4	3	**78**	77.2	**78**	**78**	frb50-23-4	4	**90**	89.2	**90**	**89.8**
frb53-24-3	2	69	68.7	**70**	**69.2**	frb50-23-5	3	78	77.9	**79**	**78**	frb50-23-5	4	90	89.7	**91**	**90.2**
frb53-24-4	2	69	68.5	**70**	**68.8**	frb53-24-1	3	83	83	**84**	**83.4**	frb53-24-1	4	97	96.3	**98**	**96.7**
frb53-24-5	2	**68**	67.1	**68**	**67.3**	frb53-24-2	3	**82**	81.1	**82**	**81.5**	frb53-24-3	4	95	94.2	**96**	**94.7**
frb56-25-1	2	**74**	73.2	**74**	**73.9**	frb53-24-3	3	82	81.5	**83**	**82.1**	frb53-24-4	4	95	93.8	**96**	**94.4**
frb56-25-2	2	74	73.1	**75**	**73.9**	frb53-24-4	3	82	81.3	**83**	**82**	frb53-24-5	4	92	91.8	**93**	**92.3**
frb56-25-3	2	**73**	72.1	**73**	**72.5**	frb53-24-5	3	80	79.6	**82**	**80.2**	frb56-25-1	4	101	100.8	**103**	**101.7**
frb56-25-4	2	**73**	**72.2**	**73**	72.1	frb56-25-1	3	88	87.2	**89**	**88**	frb56-25-2	4	100	99.9	**101**	**100.3**
frb56-25-5	2	72	72	**73**	**72.1**	frb56-25-2	3	**88**	86.6	**88**	**87.5**	frb56-25-3	4	**99**	98.1	**99**	**98.5**
frb59-26-1	2	77	76.7	**78**	**77.3**	frb56-25-3	3	86	85.3	**87**	**85.8**	frb56-25-4	4	98	97.6	**99**	**98.1**
frb59-26-2	2	**78**	76.9	**78**	**77.1**	frb56-25-4	3	**87**	**85.4**	86	85.2	frb56-25-5	4	**99**	98.2	**99**	**98.4**
frb59-26-3	2	76	75.5	**77**	**75.9**	frb56-25-5	3	**86**	85.2	**86**	**85.5**	frb59-26-1	4	105	104.6	**108**	**105.5**
frb59-26-4	2	**77**	75.9	**77**	**76.5**	frb59-26-1	3	91	90.8	**92**	**91.3**	frb59-26-2	4	**105**	104.4	**105**	**104.7**
frb59-26-5	2	76	75.7	**77**	**76.4**	frb59-26-2	3	91	90.7	**92**	**91.2**	frb59-26-4	4	**105**	103.5	**105**	**103.8**
gen400_p0.9_65	2	73	72	**74**	**73.1**	frb59-26-3	3	**91**	90.2	**91**	**90.6**	frb59-26-5	4	**105**	103.3	**105**	**103.8**
gen400_p0.9_75	2	**79**	78.2	**79**	**79**	frb59-26-4	3	**91**	90	**91**	**90.3**	keller6	4	106	104.4	**113**	**107.9**
MANN_a81	2	**2162**	2161.7	**2162**	**2161.8**	frb59-26-5	3	90	89.3	**91**	**90**						

Table 5 Experimental results of FD-TS and FD-TS-GDI on sparse instances

Instances	k	FD-TS		FD-TS-GDI		Instances	k	FD-TS		FD-TS-GDI		Instances	k	FD-TS		FD-TS-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
bn-human-BNU*_1-bg	2	**214**	**210.2**	**214**	208.8	soc-orkut-dir	3	**65**	63.5	**65**	**64**	soc-orkut	4	**62**	**62**	**62**	61.8
dbpedia-link	2	**36**	**35.6**	**36**	35.3	twitter_mpi	3	**173**	160.4	**173**	**173**	soc-orkut-dir	4	70	67.5	**71**	**68.4**
soc-orkut	2	**52**	51.2	**52**	**51.5**	web-baidu-baike	3	**33**	**27.8**	25	22.7	soc-sinaweibo	4	**62**	56.4	**62**	**62**
soc-orkut-dir	2	**58**	57.1	**58**	**57.8**	web-wikipedia-growth	3	**33**	**31.2**	**33**	31.1	web-baidu-baike	4	**33**	**25.2**	**33**	23.9
web-baidu-baike	2	**33**	**24.5**	**33**	24	wikipedia_link_en	3	**48**	19.5	**48**	**21**	web-wikipedia-growth	4	34	**33.7**	**35**	32.7
wikipedia_link_en	2	**46**	27.9	**46**	**32.7**	bn-human-BNU*_1-bg	4	**236**	235	**236**	**235.4**	wikipedia_link_en	4	48	**24.3**	**50**	22.7
bn-human-BNU*_1-bg	3	**226**	224.3	**226**	**224.7**	bn-human-BNU*_2-bg	4	**204**	200.6	**204**	**202.3**						
soc-orkut	3	**59**	57	**59**	**58.2**	dbpedia-link	4	**40**	**39.6**	**40**	39						

Table 6 Experimental results of DCCplex and DCCplex-GDI on classic instances

Instances	k	DCCplex		DCCplex-GDI		Instances	k	DCCplex		DCCplex-GDI		Instances	k	DCCplex		DCCplex-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
brock400_4	2	**33**	**32.8**	**33**	32.6	C1000.9	3	95	95	**96**	**95.2**	brock800_3	4	**34**	33.9	**34**	**34**
brock800_4	2	**26**	25.4	**26**	**25.8**	C2000.5	3	**23**	22.1	**23**	**22.3**	C1000.9	4	**109**	107.9	**109**	**108.1**
C1000.9	2	**82**	81.8	**82**	**82**	C2000.9	3	108	107.3	**109**	**107.9**	C2000.9	4	**122**	121.2	**122**	**121.3**
C2000.5	2	**20**	19.6	**20**	**19.8**	C4000.5	3	**24**	23.6	**24**	**23.9**	C4000.5	4	26	26	**27**	**26.6**
C2000.9	2	93	92.2	**94**	**92.7**	frb100-40	3	**156**	153.9	**156**	**154.4**	DSJC1000.5	4	**24**	23.8	**24**	**24**
C4000.5	2	**21**	20.6	**21**	**20.8**	frb50-23-1	3	**79**	78.7	**79**	**79**	frb100-40	4	178	176.3	**179**	**176.9**
frb100-40	2	128	127.5	**129**	**128.2**	frb50-23-2	3	**79**	78.8	**79**	**79**	frb50-23-1	4	**92**	91.4	**92**	**91.5**
frb50-23-1	2	66	66	**67**	**66.1**	frb50-23-3	3	**76**	75.8	**76**	**76**	frb50-23-2	4	**91**	90.7	**91**	**90.9**
frb50-23-5	2	66	66	**67**	**66.4**	frb50-23-4	3	**79**	78.8	**79**	**78.9**	frb50-23-3	4	**88**	87.1	**88**	**87.2**
frb53-24-1	2	**71**	**70.9**	**71**	70.8	frb50-23-5	3	79	78.7	**80**	**79.2**	frb50-23-4	4	**91**	90.7	**91**	**90.9**
frb53-24-2	2	**70**	69.9	**70**	**70**	frb53-24-1	3	**85**	84.1	**85**	**84.4**	frb50-23-5	4	**92**	91.2	**92**	**91.4**
frb53-24-3	2	**70**	69.9	**70**	**70**	frb53-24-2	3	**83**	82	**83**	**82.3**	frb53-24-1	4	98	97.9	**99**	**98**
frb53-24-4	2	**69**	68.9	**69**	**69**	frb53-24-3	3	**83**	82.9	**83**	**83**	frb53-24-2	4	**95**	94.6	**95**	**94.8**
frb56-25-1	2	**75**	74.1	**75**	**74.3**	frb53-24-4	3	83	82.8	**84**	**83.1**	frb53-24-3	4	**97**	96.2	**97**	**96.3**
frb56-25-2	2	**75**	74.4	**75**	**74.5**	frb53-24-5	3	**82**	81	**82**	**81.3**	frb53-24-4	4	**96**	95.3	**96**	**95.6**
frb56-25-3	2	**74**	73.1	**74**	**73.5**	frb56-25-1	3	**89**	88.9	**89**	**89**	frb53-24-5	4	**94**	93.3	**94**	**93.6**
frb56-25-4	2	73	73	**74**	**73.1**	frb56-25-2	3	**89**	88.4	**89**	**88.7**	frb56-25-2	4	**102**	101.6	**102**	**102**
frb56-25-5	2	**74**	73.1	**74**	**73.2**	frb56-25-3	3	**88**	86.9	**88**	**87.3**	frb56-25-3	4	**101**	100.1	**101**	**100.4**
frb59-26-2	2	**79**	78	**79**	**78.4**	frb56-25-4	3	**87**	86.3	**87**	**86.5**	frb56-25-5	4	**101**	99.9	**101**	**100.2**
frb59-26-3	2	77	76.9	**78**	**77.2**	frb56-25-5	3	**87**	86.7	**87**	**86.9**	frb59-26-1	4	107	106.8	**108**	**107**
frb59-26-4	2	**78**	77.1	**78**	**77.6**	frb59-26-1	3	**93**	92.3	**93**	**92.4**	frb59-26-2	4	**108**	106.8	**108**	**107**
frb59-26-5	2	**78**	77.8	**78**	**77.9**	frb59-26-3	3	**93**	91.8	**93**	**92**	frb59-26-3	4	106	105.9	**107**	**106.2**
gen400_p0.9_65	2	**74**	**73.1**	73	73	frb59-26-5	3	**93**	91.7	**93**	**92.2**	frb59-26-4	4	106	105.6	**107**	**106.1**
san400_0.7_2	2	32	32	**33**	**32.1**	keller6	3	92	91.2	**93**	**93**	frb59-26-5	4	106	105.2	**107**	**105.8**
san400_0.9_1	2	**103**	102.1	**103**	**102.6**	san400_0.7_2	3	**47**	46.4	**47**	**47**	keller6	4	**117**	114.8	**117**	**115**
brock800_1	3	**30**	29.3	**30**	**29.9**	san400_0.7_3	3	38	38	**39**	**38.4**	p_hat1500-2	4	**107**	106.8	**107**	**107**
brock800_3	3	**30**	29.9	**30**	**30**	brock800_1	4	**34**	33.8	**34**	**34**	san400_0.7_3	4	**50**	49.7	**50**	**50**

Table 7 Experimental results of DCCplex and DCCplex-GDI on sparse instances

Instances	k	DCCplex		DCCplex-GDI		Instances	k	DCCplex		DCCplex-GDI		Instances	k	DCCplex		DCCplex-GDI	
		max	avg	max	avg			max	avg	max	avg			max	avg	max	avg
bn-human-BNU*_1-bg	2	**214**	208.2	**214**	**210.3**	bn-human-BNU*_1-bg	3	**226**	221.6	**226**	**223.1**	bn-human-BNU*_2-bg	4	**204**	200.6	**204**	**202.3**
dbpedia-link	2	**36**	34.2	**36**	**35.2**	dbpedia-link	3	**39**	**38.4**	**39**	37.4	dbpedia-link	4	**40**	38.4	**40**	**38.9**
soc-twitter-higgs	2	**73**	**73**	**73**	70.4	soc-orkut	3	**59**	57.4	**59**	**58.1**	twitter_mpi	4	**187**	186.5	**187**	**186.9**
twitter_mpi	2	**155**	**154.9**	**155**	154.6	web-baidu-baike	3	**33**	**26.3**	**33**	22.4	web-wikipedia-growth	4	**35**	32.9	**35**	**33.6**
web-baidu-baike	2	**33**	26.3	**33**	**26.5**	wikipedia_link_en	3	**48**	**28.5**	**48**	24.8	wikipedia_link_en	4	48	**26.9**	**50**	20.2
wikipedia_link_en	2	**46**	**26.8**	**46**	26	bn-human-BNU*_1-bg	4	**236**	233.5	**236**	**234.6**						

in 40 and 58 instances, respectively. Regarding the sparse instances, all four algorithms demonstrate similar performance by achieving the same maximal solutions in 300 out of 303 instances. However, FD-TS-GDI and DCCplex-GDI exhibit a slight advantage over their competitors in terms of both maximal solutions and average solutions.

Besides, we present time consumption of all algorithms on the instances where our algorithm and the competitors achieve the same maximal and average solution values. In detail, the average run time of NuQClq, FD-TS, and DCCplex is 26.06, 13.27, and 11.46 seconds, while the average run time of NuQClq-GDI, FD-TS-GDI, and DCCplex-GDI is 22.78, 11.69, and 8.98 seconds, respectively. Obviously, the proposed method clearly improves the time consumption of the local search algorithms for clique relaxation problems.

Moreover, we compare DCCplex-GDI to KLS [27] and CFS-RLS [28] on the respective selected benchmarks of the corresponding literature and using the same experiment settings (e.g., run times). The results is presented in Tables 8 and 9. For Table 8, both DCCplex-GDI and KLS are executed with 100 independent runs, whereas for Table 9, both DCCplex-GDI and CFS-RLS are executed with 20 independent runs, following the same experimental settings (e.g., run times) as mentioned in the corresponding literature. Remark that, we don't apply $Init_{gdi}$ to the KLS algorithm and compare it with the original KLS algorithm. This is because KLS does not follow the typical "initialize + search" framework. DCCplex-GDI finds better solutions than KLS and CFS-RLS for 18 and 19 instances, respectively, while DCCplex-GDI fails to match the solutions obtained by KLS and CFS-RLS on 14 and 0 instances. Note that DCCplex doesn't use our initialization method and it only finds 17 better and 27 worse solutions than KLS. Once again, the excellent results of DCCplex-GDI can be attributed to the power of our proposed initialization method.

4.3 Effectiveness of search and structure information

To verify the necessity of the structural information and search

Table 8 Experimental results of DCCplex-GDI and KLS on all the benchmarks

Instances	k	KLS	DCCplex-GDI	Instances	k	KLS	DCCplex-GDI	Instances	k	KLS	DCCplex-GDI
		max	max			max	max			max	max
C2000.9	2	93	**94**	frb100-40	4	177	**180**	frb100-40	5	200	**201**
frb100-40	2	129	**130**	frb50-23-4	4	**92**	91	frb50-23-2	5	103	**104**
frb53-24-5	2	**69**	68	frb59-26-5	4	**107**	106	frb50-23-5	5	103	**104**
frb100-40	3	154	**156**	C1000.9	5	**122**	121	frb53-24-1	5	111	**112**
frb50-23-3	3	**77**	76	C2000.5	5	**29**	28	frb53-24-2	5	107	**108**
frb56-25-1	3	89	**90**	C2000.9	5	**137**	136	frb53-24-5	5	105	**106**
brock800_4	4	**34**	33	C500.9	5	**104**	103	frb56-25-2	5	115	**116**
C1000.9	4	**110**	109	hamming10-2	5	515	**516**	frb59-26-1	5	121	**122**
C2000.5	4	**26**	25	hamming10-4	5	**80**	79	frb59-26-2	5	121	**122**
C4000.5	4	**28**	27	keller6	5	**126**	125	frb59-26-3	5	120	**121**
hamming10-4	4	**70**	69	p_hat1500-3	5	**165**	164				

Table 9 Experimental results of DCCplex-GDI and CFS-RLE on all the benchmarks

Instances	k	CFS-RLE	DCCplex-GDI	Instances	k	CFS-RLE	DCCplex-GDI	Instances	k	CFS-RLE	DCCplex-GDI
		max	max			max	max			max	max
brock400_3	2	30	**31**	C2000.5	3	22	**23**	keller6	4	112	**117**
brock400_4	2	31	**33**	C2000.9	3	108	**109**	san400_0.7_1	4	80	**81**
C2000.5	2	19	**20**	keller6	3	92	**93**	hamming10-2	5	512	**516**
C2000.9	2	92	**94**	C4000.5	4	26	**27**	MANN_a81	5	3030	**3240**
gen400_p0.9_75	2	79	**80**	DSJC1000.5	4	23	**24**	san400_0.5_1	5	35	**36**
san1000	2	17	**18**	hamming8-2	4	128	**130**				
san400_0.7_2	2	32	**33**	hamming10-4	4	68	**69**				

information. We compare GP_{sea} with two alternative versions: (1) GP_{sea1} employs the random selection strategy when the candidate vertices have the same best $score_m$ value, instead of our selection rule; (2) GP_{sea2} selects a vertex v_i with the biggest $score_{coh}$, breaking ties randomly, instead of our selection rule. The former version only utilizes the search information while the latter one only applies the structure information. GP_{str} is compared with two versions: (1) GP_{str1} selects a vertex and a subgroup with the biggest $score_{add}$ or $score_{del}$ values, breaking ties randomly instead of our add and delete rules; (2) GP_{str2} ignores the *GroupUpdate* procedure. GP_{str1} only considers the structure information and ignores the search information whereas GP_{str2} ignores the structure and search information obtained from *GroupUpdate*. #better and #worse denote the number of instances where our proposed methods find better and worse maximal results, respectively. As shown in Table 10, the results demonstrate that both search and structure information plays an important role in GP_{sea} and GP_{str}.

Table 10 Comparative results for our proposed methods and their modified versions with different strategies on all benchmarks

Problem	#inst	Algorithm	vs. GP_{sea1}		vs. GP_{sea2}	
			#better	#worse	#better	#worse
Classic instances						
MQCP	187	N-GDI	**5**	3	**10**	2
MKKP	363	F-GDI	**15**	9	**25**	3
MKKP	363	D-GDI	**11**	6	**15**	6
Problem	#inst	Algorithm	vs. GP_{str1}		vs. GP_{str2}	
			#better	#worse	#better	#worse
Sparse instances						
MQCP	362	N-GDI	**14**	8	**15**	5
MKKP	303	F-GDI	**2**	1	**1**	0
MKKP	303	D-GDI	**2**	1	**4**	1

5 Conclusion

In this work, we propose a group driven initialization method for clique relaxation problems. The proposed methods divide the vertices into some subgroups using the search and structure information and then employ the obtained group information to help the algorithm generate an initial solution. We conduct extensive benchmarks to evaluate the performance of the proposed initialization methods and Results show that the proposed initialization method outperforms a common general initialization method for clique relaxation problems.

In the future, to enhance the effectiveness of the proposed method, we intend to incorporate new ideas [43,44]. One such idea is the utilization of Hash functions to record local optimal solutions to improve the exploitation of the search procedure. Additionally, we aim to identify the cohesive structure of the given graph through a simpler and more efficient approach. Furthermore, we intend to enhance the proposed method by incorporating a forgetting mechanism. This mechanism will enable the algorithm to concentrate more on recent search information.

Acknowledgement This work was supported by the National Natural Science Foundation of China (Grant Nos. 61806050, 61976050), CCF Huawei Hu Yanglin Foundation Theoretical Computer Science Project (CCF-HuaweiLK2023001), the Fundamental Research Funds for the Central Universities (2412022ZD015, 2412023YQ003), and Jilin Science and Technology Department (YDZJ202201ZYTS412, 20240602005RC).

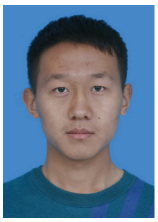
Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

References

1. Malod-Dognin N, Andonov R, Yanev N. Maximum cliques in protein

structure comparison. In: Proceedings of the 9th International Symposium on Experimental Algorithms. 2010, 106−117
2. Hao J K. Memetic algorithms in discrete optimization. In: Neri F, Cotta C, Moscato P, eds. Handbook of Memetic Algorithms. Berlin: Springer, 2012, 73−94
3. Alduaiji N, Datta A, Li J. Influence propagation model for clique-based community detection in social networks. IEEE Transactions on Computational Social Systems, 2018, 5(2): 563−575
4. Ouyang G, Dey D K, Zhang P. Clique-based method for social network clustering. Journal of Classification, 2020, 37(1): 254−274
5. Shahinpour S, Butenko S. Algorithms for the maximum *k*-club problem in graphs. Journal of Combinatorial Optimization, 2013, 26(3): 520−554
6. Jiang H, Xu F, Zheng Z, Wang B, Zhou W. A refined upper bound and inprocessing for the maximum k-plex problem. In: Proceedings of the 32nd International Joint Conference on Artificial Intelligence. 2023, 5613−5621
7. Chen J, Cai S, Pan S, Wang Y, Lin Q, Zhao M, Yin M. NuQClq: An effective local search algorithm for maximum quasi-clique problem. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. 2021, 12258−12266
8. Gao J, Xu Z, Li R, Yin M. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence. 2022, 10174−10183
9. Balasundaram B. Graph theoretic generalizations of clique: optimization and extensions. Texas A&M University, Dissertation, 2007
10. Kempe D, Kleinberg J, Tardos É. Maximizing the spread of influence through a social network. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2003, 137−146
11. Terveen L, Hill W, Amento B. Constructing, organizing, and visualizing collections of topically related web resources. ACM Transactions on Computer-Human Interaction, 1999, 6(1): 67−94
12. Yu H, Paccanaro A, Trifonov V, Gerstein M. Predicting interactions in protein networks by completing defective cliques. Bioinformatics, 2006, 22(7): 823−829
13. Bandyopadhyay S, Bhattacharyya M. Mining the largest quasi-clique in human protein interactome. Nature Precedings, 2012
14. Yannakakis M. Node-and edge-deletion NP-complete problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing. 1978, 253−264
15. Bourjolly J M, Laporte G, Pesant G. An exact algorithm for the maximum *k*-club problem in an undirected graph. European Journal of Operational Research, 2002, 138(1): 21−28
16. Balasundaram B, Butenko S, Hicks I V. Clique relaxations in social network analysis: the maximum *k*-plex problem. Operations Research, 2011, 59(1): 133−142
17. Pattillo J, Veremyev A, Butenko S, Boginski V. On the maximum quasi-clique problem. Discrete Applied Mathematics, 2013, 161(1−2): 244−257
18. Peng Y, Xu Y, Zhao H, Zhou Z, Han H. Most similar maximal clique query on large graphs. Frontiers of Computer Science, 2020, 14(3): 143601
19. Wang Z, Zhou Y, Luo C, Xiao M. A fast maximum *k*-plex algorithm parameterized by the degeneracy gap. In: Proceedings of the 32nd International Joint Conference on Artificial Intelligence. 2023, 5648−5656
20. Miao Z, Balasundaram B. An ellipsoidal bounding scheme for the quasi-clique number of a graph. INFORMS Journal on Computing, 2020, 32(3): 763−778
21. Daemi N, Borrero J S, Balasundaram B. Interdicting low-diameter cohesive subgroups in large-scale social networks. INFORMS Journal on Optimization, 2022, 4(3): 304−325
22. Almeida M T, Carvalho F D. Two-phase heuristics for the *k*-club problem. Computers & Operations Research, 2014, 52: 94−104
23. Moradi E, Balasundaram B. Finding a maximum *k*-club using the *k*-clique formulation and canonical hypercube cuts. Optimization Letters, 2018, 12(8): 1947−1957
24. Chen J, Wang Y, Cai S, Yin M, Zhou Y, Wu J. NukCP: an improved local search algorithm for maximum k-club problem. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence. 2022, 10146−10155
25. Zhou Y, Hao J K. Frequency-driven tabu search for the maximum *s*-plex problem. Computers & Operations Research, 2017, 86: 65−78
26. Chen P, Wan H, Cai S, Li J, Chen H. Local search with dynamic-threshold configuration checking and incremental neighborhood updating for maximum k-plex problem. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. 2020, 2343−2350
27. Pullan W. Local search for the maximum *k*-plex problem. Journal of Heuristics, 2021, 27(3): 303−324
28. Jin Y, Drake J H, He K, Benlic U. Reinforcement learning based coarse-to-fine search for the maximum *k*-plex problem. Applied Soft Computing, 2022, 131: 109758
29. Djeddi Y, Haddadene H A, Belacel N. An extension of adaptive multi-start tabu search for the maximum quasi-clique problem. Computers & Industrial Engineering, 2019, 132: 280−292
30. Zhou Q, Benlic U, Wu Q. An opposition-based memetic algorithm for the maximum quasi-clique problem. European Journal of Operational Research, 2020, 286(1): 63−83
31. Pinto B Q, Ribeiro C C, Riveaux J A, Rosseti I. A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. RAIRO-Operations Research, 2021, 55: S741−S763
32. Lipowski A, Lipowska D. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and Its Applications, 2012, 391(6): 2193−2196
33. Wu Q, Hao J K. A review on algorithms for maximum clique problems. European Journal of Operational Research, 2015, 242(3): 693−709
34. Cai S, Lin J, Luo C. Finding a small vertex cover in massive sparse graphs: construct, local search, and preprocess. Journal of Artificial Intelligence Research, 2017, 59: 463−494
35. Newman M E J, Girvan M. Finding and evaluating community structure in networks. Physical Review E, 2004, 69(2): 026113
36. Blondel V D, Guillaume J L, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008, 2008(10): P10008
37. López-Ibáñez M, Dubois-Lacoste J, Cáceres L P, Birattari M, Stützle T. The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives, 2016, 3: 43−58
38. Johnson D S, Trick M A. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11−13, 1993. Boston: American Mathematical Society, 1996
39. Xu K, Boussemart F, Hemery F, Lecoutre C. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. Artificial Intelligence, 2007, 171(8-9): 514−534
40. Davis T A, Hu Y. The university of Florida sparse matrix collection. ACM Transactions on Mathematical Software, 2011, 38(1): 1
41. Rossi R A, Ahmed N K. The network data repository with interactive graph analytics and visualization. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. 2015, 4292−4293
42. Wang Y, Cai S, Chen J, Yin M. SCCWalk: an efficient local search algorithm and its improvements for maximum weight clique problem. Artificial Intelligence, 2020, 280: 103230

43. Jiang L, Ouyang D, Zhang Q, Zhang L. DeciLS-PBO: an effective local search method for pseudo-Boolean optimization. Frontiers of Computer Science, 2024, 18(2): 182326
44. Pan S, Ma Y, Wang Y, Zhou Z, Ji J, Yin M, Hu S. An improved master-apprentice evolutionary algorithm for minimum independent dominating set problem. Frontiers of Computer Science, 2023, 17(4): 174326



Rui Sun received the BS and MS degrees from the Software College, Jilin University, China in 2019 and 2022, respectively. His current research interests include local search, combinatorial optimization, propositional satisfiability problems.

Yiyuan Wang is an associate professor at School of Computer Science and Information Technology, Northeast Normal University, China. He received his PhD degree from Jilin University, China. His research interests include heuristic search, local search, algorithmic design, combinatorial optimization.

Minghao Yin is a professor at School of Computer Science and Information Technology, Northeast Normal University, China. He received his PhD degree in Computer Software and Theory from Jilin University, China. His research interests include heuristic search, data mining, and combinatorial optimization.