Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)

Nukplex: An Efficient Local Search Algorithm for Maximum K-Plex Problem

Rui Sun[1] , **Yiyuan Wang**[1,2*] , **Shimao Wang**[1] , **Hui Li**[1] , **Ximing Li**[3,4] , **Minghao Yin**[1,2,*]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
[3]College of Computer Science and Technology, Jilin University, China
[4]Key Laboratory of Symbolic Computation and Knowledge Engineering of MOE, Jilin University, China
ruisun@nenu.edu.cn, wangyy912@nenu.edu.cn, wangsm928@nenu.edu.cn, lihui@nenu.edu.cn, liximing86@gmail.com, ymh@nenu.edu.cn

Abstract

The maximum k-plex problem (MKPP) is a significant relaxation version of the maximum clique problem with extensive applications. Recently, lots of researchers have proposed many heuristic algorithms based on various methods to solve the MKPP. In this work, to further improve the performance of solving the MKPP, we propose an efficient local search algorithm based on three main ideas. First, we propose a relaxed bounded configuration checking strategy that considers two kinds of historical searching information to relax the restricted strength of configuration checking and the forbidden condition of candidate vertices for the Add operation, respectively. Second, we present a novel solution information-based vertex selection strategy based on two kinds of solution information to select high-quality candidate vertices. Third, we define the solution core and then introduce a core-based perturbation strategy to help the algorithm jump out of local optima. The experimental results show that the proposed algorithm significantly outperforms the state-of-the-art MKPP algorithms in almost all the instances.

1 Introduction

Cohesive subgroups are subsets of actors among whom there are relatively strong, direct, intense, frequent, or positive ties. In graph theory, the model of cohesive subgroups is usually used to analyze some real-world applications, especially in social network analysis [Wasserman and Faust, 1994]. The earliest mathematical model of a cohesive subgroup was the clique. Formally, a clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. The ideal characteristic of the clique is the result of requiring a complete subgraph. However, this kind of model is overly restrictive in practice [Seidman and Foster, 1978]. Thus, to deal with real-world scenarios, researchers have proposed several clique relaxation problems, such as k-plex [Chen *et al.*, 2020], k-club [Shahinpour and Butenko, 2013] and k-quasi-clique [Chen *et al.*, 2021], which can relax specific properties of the clique to overcome its modeling limitations. In this paper, we focus on studying the maximum k-plex problem (MKPP), which has been successfully applied in social network analysis [Kondo *et al.*, 2012; Pattillo *et al.*, 2013; Xiao *et al.*, 2017], especially in community detection [Conte *et al.*, 2018; Zhu *et al.*, 2020]. For example, when detecting potential money laundering crimes, a common approach is to construct a graph representation of wire transfer databases, where each edge represents the flow of money [Balasundaram, 2007]. In this application, the primary focus is often on the edge information, and the flow of money information may be unclear or incomplete. Therefore, the aforementioned application can be formulated as k-plex and effectively addressed using k-plex algorithms.

Given a graph $G = (V, E)$ and a fixed positive integer k, a k-plex S is a subset of vertices that each vertex is at least adjacent with $|S| - k$ vertices. The MKPP aims to find a k-plex with the maximum size. As is known, the MKPP has been proven to be an NP-hard problem [Balasundaram *et al.*, 2011]. Therefore, solving this problem is not an easy task. Both exact and heuristic algorithms have been studied.

The exact algorithms for the MKPP mainly comprise three categories: integer programming based algorithms [Balasundaram *et al.*, 2011], branch and bound algorithms [McClosky *et al.*, 2012; Moser *et al.*, 2012; Xiao *et al.*, 2017; Gao *et al.*, 2018; Wu *et al.*, 2019; Zhou *et al.*, 2021; Jiang *et al.*, 2021; Chang *et al.*, 2022; Jiang *et al.*, 2023; Wang *et al.*, 2023] and Russian doll search [Trukhanov *et al.*, 2013; Shirokikh, 2013; Gschwind *et al.*, 2018]. These exact algorithms can guarantee the optimality of their solutions, but they can hardly perform very well for some hard instances within a reasonable time. To settle this issue, researchers usually resort to designing heuristic algorithms to obtain a good solution.

Generally speaking, the heuristic algorithms for the MKPP can be divided into hybrid metaheuristic algorithms [Gujjula *et al.*, 2014; Miao and Balasundaram, 2017], learning-based algorithm [Jin *et al.*, 2022], and local search algorithms [Zhou and Hao, 2017; Chen *et al.*, 2020; Pullan, 2021]. According to the literature, the current best heuristic algorithms for the MKPP are DCCplex [Chen *et al.*, 2020] and KLS [Pullan, 2021].

Motivated to contribute to further improving the performance of solving the MKPP, in this study, we present an effi-

*Corresponding author

cient local search algorithm named Nukplex based on three novel ideas. First, we propose a new variant of the configuration checking (CC) strategy. CC is a commonly used strategy for the cycling problem and has been used in various combinational optimization problems [Cai *et al.*, 2011; Wang *et al.*, 2020; Chen *et al.*, 2022; Chen *et al.*, 2023]. Considering the characteristic of the MKPP and the three basic operations of local search algorithms for the MKPP, we propose a relaxed bounded configuration checking strategy (RBCC). In the proposed strategy, we relax the restricted strength of configuration checking by considering the vertices' frequency information and allow more candidate added vertex to be selected by using a traditional tabu mechanism [Glover, 1989] during the search procedure.

Second, different from the random selection method and the structure-based selection method adopted by previous local search algorithms [Zhou and Hao, 2017; Chen *et al.*, 2020; Pullan, 2021], we propose a novel solution information-based vertex selection strategy based on two novel scoring functions. The first scoring function takes the structure information of a given instance (i.e., degree value) and the current solution into account, whereas the second scoring function considers the historical information of the search procedure (i.e., local best solution). Note that these scoring functions measure the gain of adding a vertex from different perspectives.

Third, a core-based perturbation strategy is proposed to diversify the search direction. It extracts a cohesive subset from a candidate solution based on the definition of the solution core and then adds several vertices to extend this subset into a new perturbed solution until the perturbed solution differs from the original solution to a certain extent.

To evaluate the effectiveness of the proposed algorithm, extensive experiments are carried out on the benchmarks adopted by the previous literature for the MKPP. Experimental results show that the proposed algorithm outperforms the state-of-the-art heuristic and exact algorithms in almost all benchmarks, and the proposed strategies play an essential role in the excellent performance of the proposed algorithm.

In the next section, we introduce some necessary background knowledge. After that, we present our proposed three ideas and our algorithm. Experimental results are shown in Section 7. Finally, we make conclusions.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph, where V is a vertex set with n vertices and E is an undirected edge set with m edges. Each edge $e \in E$ has two endpoints, denoted as $e = \{v, u\}$. The density of graph G is defined as $dens(G) = \frac{|E|}{\binom{|V|}{2}}$. For a vertex $v \in V$, the set of its neighbors is defined as $N(v) = \{u \in V \mid \{u, v\} \in E\}$ and its degree $deg(v)$ is defined as the number of its neighbors, denoted as $deg(v) = |N(v)|$. For a vertex set $S \subseteq V$, $N(S) = \bigcup_{v \in S} N(v) \setminus S$. The induced subgraph $G[S] = (S, E_S)$ is a subgraph of G whose vertex set is S and whose edge set includes all the edges in E that have both endpoints in S.

Given a graph G and a positive integer k, a subset S of V is a k-plex such that $|N(v) \cap S| \geq |S| - k$ for each $v \in S$. Obviously, each vertex of a k-plex S must be adjacent to at least $|S| - k$ vertices in the subgraph $G[S]$. The maximum k-plex problem (MKPP) is to find a k-plex with the most vertices.

During the search process, we use S to denote the feasible candidate solution (i.e., being a k-plex). For a vertex $v \in S$, if $|N(v) \cap S| = |S| - k$, v is called a saturated vertex. We define $C(S)$ as the set of all saturated vertices in S. Local search algorithms for MKPP usually modify the feasible candidate solution S iteratively through three basic operations, including Add, Swap, and Drop [Zhou and Hao, 2017; Chen *et al.*, 2020]. With the current solution denoted by S, we introduce three candidate sets for each of the three operations.

- $AddSet(S) = \{v \in N(S) \mid |N(v) \cap S| > |S| - k, C(S) \setminus N(v) = \emptyset\}$
- $SwapSet(S) = \{v \in N(S) \mid |N(v) \cap S| \geq |S| - k, |C(S) \setminus N(v)| = 1\} \cup \{v \in N(S) \mid |N(v) \cap S| = |S| - k, |C(S) \setminus N(v)| = \emptyset\}$
- $DropSet(S) = S$

$AddSet(S)$ refers to adding a vertex $v \in N(S)$ into a candidate solution, provided that the vertex is adjacent to more than $|S| - k$ neighbors and any saturated vertex is excluded in the set of v's non-neighbors. It is obvious that the operation Add can increase the cardinality of the solution by one, and thus this operation always leads to a better solution. $SwapSet(S)$ contains a pair of vertices which is eligible for exchanging only if it satisfies one of the following two conditions: (i) v is adjacent to at least $|S| - k$ vertices in S and the set of v's non-neighbors includes only one saturated vertex; (ii) v is adjacent to exactly $|S| - k$ vertices in S and the set of v's non-neighbors does not contain any saturated vertex. During the search process, the algorithm will add a vertex $v \in SwapSet(S)$ into the candidate solution and randomly remove one vertex from the candidate solution, ensuring that the candidate solution is still a k-plex. Specifically, if $|N(v) \cap S| \geq |S| - k$ and $|C(S) \setminus N(v)| = 1$, the saturated vertex connected to v will be removed. Otherwise, if $|N(v) \cap S| = |S| - k$ and $|C(S) \setminus N(v)| = \emptyset$, a vertex in $S \setminus N(v)$ will be randomly selected and then removed. $DropSet(S)$ includes all the vertices in S. The Drop operation is considered only when Add and Swap operations are unavailable.

3 Relaxed Bounded Configuration Checking Strategy

Configuration checking (CC) as a diversification strategy has been widely used to avoid the cycling problem in local search [Cai *et al.*, 2011]. It works as follows: for $v \notin S$, if its configuration has not changed since v's last removal from S, it is forbidden to be added back to S. Typically, the configuration of a vertex refers to the state of its neighboring vertices. The CC strategy is usually implemented with an array named $conf$, where $conf(v) = 1$ means v is allowed to be added to the candidate solution, and $conf(v) = 0$ means it should not be added.

3.1 Review of CC Strategies for Clique Relaxation Problems

Recently, three versions of CC have been proposed for clique relaxation problems, including dynamic-threshold configuration checking (DCC) [2020] for the MKPP, bounded configuration checking (BoundedCC) [2021] for the maximum quasi-clique problem, and stratified threshold configuration checking (STCC) [2022] for the maximum k-club problem. The CC strategies mentioned above all maintain an integer threshold $thred$ to control the forbidding strength. Only when $conf(v) \geq thred(v)$, v is allowed to be added back to S.

BoundedCC and STCC can be considered as two variants of DCC. (1) BoundedCC sets an upper bound on the threshold of the DCC condition, which can avoid the frequently operated vertices forbidden for a long time. In detail, when $thred(v)$ reaches the predefined upper bound denoted as ub_thre, it will be reset to 1. (2) STCC considers the specific feature of the maximum k-club problem and more details can be seen [Chen *et al.*, 2022].

3.2 The RBCC Strategy

When the threshold of vertex v (i.e., $thred(v)$) reaches the upper bound ub_thre, indicating that the vertex has been forbidden for a period of time, BoundedCC simply resets its threshold value to 1. According to the preliminary experiment, this updating way would mislead the search by forbidding some promising candidate vertices. To address it, we design a novel updating rule by taking the vertices' frequency into account, which can be considered as a kind of search information indicating the accumulative effectiveness of the search on each vertex.

In detail, each vertex $v \in V$ has an additional property, frequency, denoted by $freq(v)$. The $freq$ value of each vertex is initialized to 1. During the search process, the $freq$ value of each vertex is used to record the total number of times that the vertex has been added or removed. Here we suppose that the algorithm adds vertex v_1 into the candidate solution and its threshold value is larger than ub_thre. Intuitively, in this case, we observe that if $freq(v_1)$ is not too high compared to some other vertices, then we should further relax its forbidden strength. To implement it, we randomly sample t vertices and put these vertices into a frequency candidate set denoted as $FreqSet = \{v_1^f, v_2^f, \ldots, v_t^f\}$ where the positions of vertices are arranged in a descending order of the frequency values. If $freq(v_1)$ is less than $freq(v^f_{\lfloor 0.8\times t \rfloor})$, we think that the CC condition of the vertex v should be relaxed, i.e., setting $thred(v_1)$ to 0.

Based on the above discussion, we modify BoundedCC into a more relaxed version, which is called relaxed bounded configuration checking (RBCC) strategy. This strategy is specified by the following four rules.

RBCC Initial Rule. At the beginning of search process, for each $v \in V$, $conf(v)$ and $thred(v)$ are initialized to 1.

RBCC Add Rule. When v is added into the candidate solution, $thred(v)$ and $conf(v')$ are increased by 1 for each $v' \in N(v)$. If $thred(v) \geq ub\_thre$, we randomly select t vertices to generate a frequency candidate set $FreqSet = \{v_1^f, v_2^f, \ldots, v_t^f\}$. There are two cases: (1) if $freq(v_1) < freq(v^f_{\lfloor 0.8\times t \rfloor})$, then $thred(v_1)$ is reset to 0; (2) Otherwise, $thred(v_1)$ is reset to 1.

RBCC Swap Rule. When u is removed and v is added into the candidate solution, $conf(u)$ is switched to 0.

RBCC Drop Rule. When removing a vertex u from the candidate solution, $conf(u)$ is set to 0.

Two Constrained Candidate Sets

From experimental observation, we learnt that in local search algorithms for MKPP, each operation (i.e., Add, Swap, and Drop) has different effects on a candidate solution. The operation Add can directly improve the quality of the candidate solution, whereas the operation Swap can be seen as a form of diversification by leading the search switch to another candidate solution near the current one. Thus, we believe the vertices in the $AddSet$ should be encouraged to be added to the candidate solution. Different from DCC, BoundedCC, and STCC that enforce a uniform CC constraint for the operations Add and Swap, we use a classic tabu mechanism [Glover, 1989] as an auxiliary way to relax the CC constraint for the operation Add. Specifically, for each vertex $v \in V$, $age(v)$ is used to record the number of steps since the last time it was removed from the candidate solution.

In the following, we employ the tabu and RBCC to redefine the candidate sets for the operations Add and Swap.

- $ConstrAddSet(S) = \{v \in AddSet(S) \mid conf(v) \geq ub\_thre\} \cup \{v \in AddSet(S) \mid age(v) > L_t\}$
- $ConstrSwapSet(S) = \{v \in SwapSet(S) \mid conf(v) \geq ub\_thre\}$

Where L_t is a parameter for the tabu mechanism.

4 A Novel Solution Information-Based Vertex Selection Strategy

In this section, we first review previous vertex selection strategies used to solve the MKPP. Then, we make use of the information of the current solution and previous generated local best solutions to measure each candidate vertex in the respective candidate sets, resulting in two novel scoring functions. Finally, we design a novel solution information-based vertex selection strategy.

4.1 Previous Vertex Selection Strategies

Before introducing our proposed scoring functions, we review two previous methods to select the added and swapped vertices, namely random-based and structure-based methods.

The random-based method is used by previous MKPP algorithms such as FD-TS [Zhou and Hao, 2017] and DCCplex [Chen *et al.*, 2020]. To overcome the cycling problem, the algorithms usually use tabu [Glover, 1989] or CC strategies [Cai *et al.*, 2011] to reduce the $AddSet$ and $SwapSet$. During the adding and swapping processes, if the corresponding reduced candidate set has any vertices, the algorithm chooses a random vertex from the candidate set.

As for the second structure-based method, such as KLS [Pullan, 2021], the algorithm also uses a tabu mechanism to reduce the candidate set and then uses a common scoring function, denoted as d_S, to select a candidate vertex. Given a current candidate solution $S \subseteq V$, the adjacency $d_S(v)$ of a vertex $v \in N(S)$ is the number of vertices in S that are connected to v, which is defined as below.

$$d_S(v) = |N(v) \cap S|$$

4.2 Current Solution Information-Based Scoring Function

It is obvious that adding a vertex with a high d_S value usually leads to few saturated vertices. Thus, in our work, we mainly consider the d_S as the selection basis to design our first scoring function. Besides, $\overline{d_S}(v) = deg(v) - d_S(v)$ represents the number of vertices in $V \setminus S$ that connects v after adding v into S. The $\overline{d_S}(v)$ value denotes the potential capacity of expanding S after adding v. In the following, we present a novel current solution information-based scoring function.

$$score_{csi}(v) = d_S(v) \times (1 + \frac{\rho \times \overline{d_S}(v)}{|V|})$$

Where ρ is a parameter whose range is from 0 to 1. On the one hand, based on the definition of $score_{csi}$, we can easily obtain that $d_S(v)$ has the greatest influence on the value of $score_{csi}$. On the other hand, because we utilize $\frac{\rho \times \overline{d_S}(v)}{|V|}$ as the coefficient of the $d_S(v)$, $\overline{d_S}$ can be viewed as the second influencing factor of $score_{csi}$ and $\frac{\rho}{|V|}$ is used to balance the influence between these two factors, i.e., d_S and $\overline{d_S}$.

The intuition underlying the $score_{csi}$ is to fully explore the information of current solutions. In detail, $score_{csi}$ considers the immediate impact (i.e., d_S) and subsequent extendibility capacity (i.e., $\overline{d_S}$) of S when adding a vertex.

4.3 Local Best Solution Information-Based Scoring Function

Local search algorithms for the MKPP usually adopt a classic restart search framework [Chen *et al.*, 2021; Pullan, 2021], which consists of two procedures: construction and search procedures. In each round, the construction procedure iteratively generates an initial solution, and the search procedure improves the initial solution and then returns a local best solution. The local best solution information-based scoring function $score_{lbs}$ is designed by utilizing the historical local optimal solutions. Its corresponding rules are given as below.

Initial Rule. At the beginning, $score_{lbs}(v)$ is initialized to 1, for every $v \in V$.

Update Rule. When the local search procedure returns a local best solution denoted as S_{lbest}, for every $v \in V$, $score_{lbs}(v)$ is increased by $d_{S_{lbest}}(v)$.

The designing of $score_{lbs}$ is inspired by the backbone structure of combinational optimization problems [Wu and Hao, 2015]. It indicates that for many optimization problems, high-quality solutions usually share some same components. The intuition evaluation criterion of the backbone is the number of times that a variable occurs in the local optimal solutions. In our work, we generalized this idea as the accumulative values of the number of connections between v and each local optimal solution, i.e., $score_{lbs}$.

4.4 Our Proposed Vertex Selection Strategy

Based on the above two scoring functions, we design two novel vertex selection rules as follows.

Add Rule. Adding one vertex $v \in ConstrAddSet(S)$ with the highest value of $score_{csi}(v)$ into S, breaking ties randomly.

Swap Rule. The vertex selection rule for $ConstrSwapSet(S)$ considers two conditions as below.

- The first case $dens(G) > \alpha$. Selecting one vertex $v \in ConstrSwapSet(S)$ with the highest value of $score_{lbs}(v)$ into S, breaking ties randomly.
- The second case $dens(G) \leq \alpha$. Selecting a vertex v with the lowest $freq(v)$ value, breaking ties randomly.

In our work, parameter α is set to 0.35 according to preliminary experiments. For the Swap operation, after adding v, we remove a vertex that is not connected to v from S. We have introduced how to select a removed vertex in Section 2.

For the Add operation, because adding a vertex can directly improve the solution quality, in order to make sure the expandability of the candidate solution after adding the vertex, we adopt a greedy manner based on the value of $score_{csi}$ to decide which vertex is a candidate added vertex. As for the Swap operation that aims to transfer the current candidate solution to its neighbor search space, instead of adopting a greedy vertex selection method, we utilize the search information to explore search spaces based on the structure information of a given graph (e.g., density). In detail, if the graph is very sparse, the search procedure is easy to fall into local optima, and thus we focus on searching for some rarely visited search spaces based on the value of $freq$. Otherwise, we think the graph is dense. We turn to visit some potential high-quality search spaces by considering the value of $score_{lbs}$.

5 A Core-Based Perturbation Strategy

Even though our proposed RBCC can help the algorithm overcome the cycling problem, this strategy cannot deal with the issue that the search process gets stuck in a search space. To address it, we propose a core-based perturbation strategy based on the definition of solution core. The proposed strategy contains shrinking and expanding phases. In the shrinking phase, the candidate solution is refined to a solution core. The expanding phase adopts a heuristic mechanism to generate a new perturbed solution by extending the solution core. First, we give the definition of the solution core.

Definition 1 (Solution Core)**.** *Given a graph $G = (V, E)$, an integer $k \geq 2$ and a k-plex solution S, a solution core of S is a subset $S_{core} \subseteq S$ such that for each $v \in S_{core}$, $|N(v) \cap S_{core}| > |S_{core}| - k$.*

Given a k-plex solution S, its solution core S_{core} is a subset of S that satisfies the constraint condition of (k-1)-plex. S_{core} is a more cohesive structure than S.

Based on the definition of solution core, we present a core-based perturbation strategy in Algorithm 1. In the shrinking phase (Lines 2–5), a solution core can be obtained by iteratively removing a vertex from the set of all saturated vertices $C(S)$ until $C(S)$ becomes empty. During the above

Algorithm 1: CorePerturb

Input: the candidate solution S
Output: the perturbation solution S
1 $TempSet := \emptyset$;
2 **while** $C(S) \neq \emptyset$ **do**
3 select a random saturated vertex $u \in C(S)$;
4 $S := S \setminus \{u\}$ and $TempSet := TempSet \cup \{u\}$;
5 update the $conf(u)$ value based on **RBCC Drop Rule**;
6 **while** $AddSet \cap TempSet \neq \emptyset$ && $AddSet \setminus TempSet \neq \emptyset$ **do**
7 **if** *with probability* p **then**
8 select a random vertex $v \in AddSet \cap TempSet$;
9 **else**
10 select a random vertex $v \in AddSet \setminus TempSet$;
11 $S := S \cup \{v\}$;
12 update the $conf$ and $thred$ values of v and its neighbors according to **RBCC Add Rule**;
13 **return** S;

phase, all removed vertices are stored in a set $TempSet$. In the expanding phase (Lines 6–12), to disturb the candidate solution S, the algorithm iteratively adds a vertex in $AddSet \cap TempSet$ with probability p and otherwise it adds a vertex in $AddSet \setminus TempSet$ until one of the above two sets becomes empty. Finally, the perturbed candidate solution is returned (Line 13).

6 The Nukplex Algorithm

Based on the above three strategies, we propose a local search algorithm Nukplex in Algorithm 2. At the beginning, a global best solution S_{best} is initialized as an empty set (Line 1). In each round, the algorithm initializes the four variables accordingly (Line 3–4). The algorithm iteratively calls the construction procedure (Line 5) and the local search procedure (Lines 6–24). Finally, the algorithm returns S_{best} (Line 29).

During the construction procedure, the algorithm iteratively selects a vertex with the smallest $freq$ value in $AddSet$ and then adds it into S until $AddSet$ becomes empty. This procedure aims to lead the search procedure to some unvisited search spaces.

During the local search procedure, S_{lbest} denotes the local best solution, $unimpr$ records the number of unimproved steps, and a parameter L denotes the search depth. In each iteration, $ConstrAddSet$ and $ConstrSwapSet$ are checked sequentially. If $ConstrAddSet$ is not empty, the algorithm adds a vertex v into S based on the add rule (Lines 7–9). Otherwise, if $ConstrSwapSet$ is not empty, the algorithm exchanges a pair of vertices based on the swap rule (Lines 11–13). If the above two sets are empty, a random vertex in S will be removed (Lines 15–16). Note that the $thre$ and $conf$ values are updated accordingly based on the corresponding RBCC rules (Lines 10, 14 and 17). The $score_{lbs}$ value of

Algorithm 2: Nukplex

Input: graph $G = (V, E)$, the *cutoff* time and parameter k
Output: a best solution S_{best} found
1 $S_{best} := \emptyset$;
2 **while do**
3 $cur\_step := unimpr := 0$;
4 initialize the $conf$ and $thred$ values of each vertex based on **RBCC Initial Rule**;
5 $S_{lbest} := S := InitConstruct(G)$;
6 **while** $cur\_step < L$ **do**
7 **if** $ConstrAddSet(S) \neq \emptyset$ **then**
8 select an added vertex v based on **Add Rule**;
9 $S := S \cup \{v\}$;
10 update the $conf$ and $thred$ values of the corresponding vertices based on **RBCC Add Rule**;
11 **else if** $ConstrSwapSet(S) \neq \emptyset$ **then**
12 select an added vertex v and a removed vertex u based on **Swap Rule**;
13 $S := S \cup \{v\} \setminus \{u\}$;
14 update the $conf$ values of the corresponding vertices based on **RBCC Swap Rule**;
15 **else**
16 select a random removed vertex u from $DropSet$ and $S := S \setminus \{u\}$;
17 update the $conf$ values of the corresponding vertices based on **RBCC Drop Rule**;
18 update the related scoring function values based on **Update Rule**;
19 **if** $|S| > |S_{lbest}|$ **then**
20 $S_{lbest} := S$ and $unimpr := 0$;
21 **if** $unimpr > k \times |S_{best}|$ && $S_{best} \neq \emptyset$ **then**
22 $S := CorePerturb(S)$ and $unimpr := 0$;
23 $cur\_step := cur\_step + 1$;
24 $unimpr := unimpr + 1$;
25 **if** $|S_{lbest}| > |S_{best}|$ **then**
26 $S_{best} := S_{lbest}$;
27 $G := Reduce(G, |S_{best}|, k)$;
28 **if** $|V| \leq |S_{best}|$ **then return** S_{best};
29 **return** S_{best};

each vertex is updated according to the information of S_{lbest} (Line 18). If S is better than S_{lbest}, the algorithm updates S_{lbest} by S and $unimpr$ is reset to 0 (Lines 19–20). Then, the algorithm checks whether the unimproved step $unimpr$ reaches a fixed value, i.e., $k \times |S_{best}|$ (Line 21). If so, the algorithm performs the core-based perturbation strategy and resets $unimpr$ to 0 (Lines 21–22). At the end of each round, both cur_step and $unimpr$ are increased by one (Lines 23–24).

After the search procedure, if a local best solution S_{lbest} is better than S_{best}, S_{best} is updated by S_{lbest} (Lines 25–26). The algorithm reduces G by calling the $Reduce$ function that deletes the vertices whose degree value is less than $|S_{best}| - (k-1)$ (Line 27). After reducing some vertices, if $S_{best} \geq |V|$, which means that the algorithm has already found an optimal solution, S_{best} is returned (Line 28).

7 Experimental Evaluation

In this section, we evaluate the performance of Nukplex[1] on a broad range of classic instances and sparse large instances. According to previous studies [Chen *et al.*, 2020; Pullan, 2021; Jin *et al.*, 2022], the best results of KLS [Pullan, 2021] and DCCplex [Chen *et al.*, 2020] totally dominated the results obtained by other MKPP heuristic algorithms. Thus, we compare Nukplex with these two algorithms. In addition, we also compare Nukplex with two state-of-the-art exact MKPP algorithms KpLeX-Gap [Wang *et al.*, 2023] and DiseMKP [Jiang *et al.*, 2023]. The source code of all algorithms are kindly provided by the authors. All algorithms were implemented in C++ and compiled by g++ with '-O3' option. For DCCplex and KLS, we set the same parameters as described in corresponding literature and optimize these parameters for newly added instances. All experiments are run on Intel Xeon Gold 6238 CPU @ 2.10GHz CPU with 512GB RAM under CentOS 7.9.

Our computational assessment was conducted on classic instances and sparse large instances for $k = 2, 3, 4, 5$, which were also reported in the previous MKPP studies [Zhou and Hao, 2017; Chen *et al.*, 2020; Pullan, 2021; Jin *et al.*, 2022]. First, we considered two classical benchmarks, including 80 and 41 graphs from DIMACS [Johnson and Trick, 1996] and BHOSLIB [Xu *et al.*, 2007], respectively. We also evaluated Nukplex on sparse large instances, including 67 graphs from the Stanford Large Network Dataset Collection (SNAP)[2] and the 10th DIMACS implementation challenge (DIMACS10)[3] and 42 graphs from the Network Data Repository (NDR) [Rossi and Ahmed, 2015]. Note that the same graph with different k is considered as the different instances. In total, we select 920 tested instances. For each instance, all heuristic algorithms are executed 100 times with the random seeds from 1 to 100. The cutoff time for classic instances is set to 1000 seconds. As for the sparse large instances, we report the results under the cutoff time of 100 and 1000 seconds, respectively. For two exact algorithms KpLeX-Gap and DiseMKP, the cutoff time is set to 1800 seconds according to the literature. In some cases, the size of solution obtained by an exact algorithm within a predefined time limit may be equal to the size of optimal solution, but its optimality has not been formally verified. So we compare Nukplex with the best solution achieved by two exact algorithms within 1800 seconds.

According to our preliminary experiments by using the automatic configuration tool irace [López-Ibáñez *et al.*, 2016], Table 1 shows the selected parameter values. Specifically, since the two benchmarks have different scales, we built a training set and randomly selected 10 instances from the corresponding tested benchmark. For each instance, we set k to 4 and 5, respectively. The tuning process is given a budget of 8000 runs for the training set with a time budget of 1000 seconds per run. The results have indicated that our algorithm is not sensitive to six of these parameters. However, we have found that the value of parameter p affects our algorithm's performance due to its involvement in the vertex selection phase of the perturbation strategy. Further experiments showed that the best solution achieved using the optimal parameter settings is on average 0.77% larger than the solutions obtained from other parameter combinations.

Parameter	Range	Final value
L	$\{3000, 4000, 5000\}$	4000
t	$\{40, 50, 60, 70\}$	50
L_t	$\{3, 4, 5\}$	4
ub_thre	$\{2, 3, 4, 5\}$	3
ρ	$\{0, 0.1, 0.2, 0.3\}$	0.2
p	$\{0.4, 0.5, 0.6, 0.7\}$	0.5

Table 1: Tuned parameters of our proposed algorithm.

7.1 Results on Classic and Sparse Large Instances

Table 2 summarizes the experimental results. Observed from Table 2, Nukplex achieves the best solution on almost all instances. We will discuss the comparisons against MKPP heuristic algorithms and MKPP exact algorithms separately as follows.

Comparisons against MKPP heuristic algorithms. For DIMACS and BHOSLIB instances, Nukplex outperforms KLS and DCCplex for 34 and 49 instances, while it is defeated for only 3 and 1 instances, respectively. Furthermore, for instances where Nukplex and a corresponding algorithm achieve the same best solution, Nukplex finds better average solutions than KLS and DCCplex for 86 instances and 143 instances. Considering that Nukplex and a corresponding competitor obtain the same maximal and average values, the average run time of Nukplex is 3.49 (or 0.99) seconds while the average run time of KLS (or DCCplex) is 7.18 (or 6.41) seconds. In addition, we adopt the performance profile [Dolan and Moré, 2002] to evaluate the time consumption of the three algorithms when they obtain the same solution quality. It shows the probability $P(r \leq \tau)$ of obtaining the best solution in a time that at most a factor τ slower than the fastest algorithm. When $\tau = 1$, $P(r \leq \tau)$ denotes the probability that an algorithm is the fastest. As shown in Figure 1 (a), Nukplex consistently performs better than two competitors, which demonstrates the effectiveness of our proposed algorithm for the classic instances.

The best solution achieved by Nukplex totally dominates DCCplex and KLS on the sparse large instances. Specifically, Nukplex obtains better solutions than DCCplex and KLS for 8 and 6 instances, respectively. Observed from the results in Table 2, the performance difference among the three algorithms is not obvious. In detail, for the instances where our proposed algorithm and a corresponding gets the same max-

[1]Source code and supplementary materials are available at https://github.com/yiyuanwang1988/Nukplex

[2]http://snap.stanford.edu/data

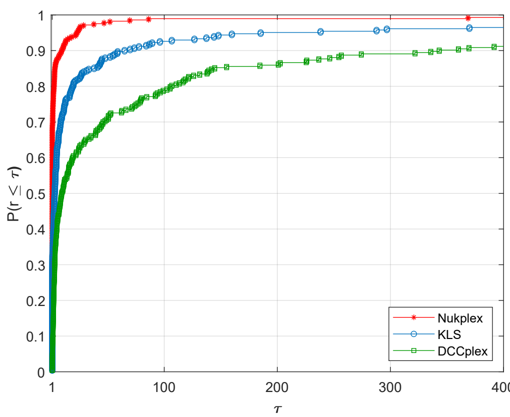
[3]https://www.cc.gatech.edu/dimacs10/

Benchmark	#inst.	Nukplex #max(#avg)		KLS #max(#avg)		DCCplex #max(#avg)		DiseMKP #max	KpLeX-Gap #max
classic	484	ct=1000s **480(466)**		ct=1000s 449(361)		ct=1000s 433(288)		ct=1800s 123	ct=1800s 87
large sparse	436	ct=1000s **436(436)**	ct=100s 434(429)	ct=1000s 435(434)	ct=100s 431(426)	ct=1000s 434(432)	ct=100s 430(412)	ct=1800s 406	ct=1800s 418
#total	920	**916(902)**	914(895)	884(795)	880(787)	867(720)	863(640)	529	506

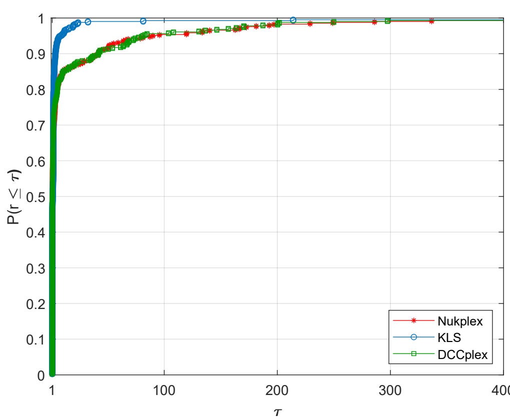
Table 2: Comparative results on the two benchmarks are presented, where ct represents the cutoff time for each algorithm. #max and #avg denotes the number of maximal and average results found by each algorithm, respectively.

Benchmark	#inst.	vs. Nukp1 #bet(#wor)	vs. Nukp2 #bet(#wor)	vs. Nukp3 #bet(#wor)	vs. Nukp4 #bet(#wor)	vs. Nukp5 #bet(#wor)	vs. Nukp6 #bet(#wor)	vs. Nukp7 #bet(#wor)
classic(1000s)	484	**26**(2)	**27**(3)	**24**(2)	**12**(3)	**7**(2)	**34**(3)	**11**(2)
large sparse(100s)	436	**1**(0)	**1**(0)	**1**(0)	0(0)	**1**(0)	0(0)	**2**(0)
large sparse(1000s)	436	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)

Table 3: Comparing Nukplex with 7 modified versions. #bet and #wor represent respectively the number of instances where Nukplex achieves better and worse maximal solutions.



(a) Classic instances



(b) Large sparse instances

Figure 1: Performance profiles for Nukplex and two competitors for reaching the best solution on all the benchmarks.

imal and average values, the average run time of Nukplex is 2.12 (or 2.03) seconds, while the average run time of KLS (or DCCplex) is 2.29 (or 2.16) seconds. The performance profile plot of the three algorithms in Figure 1 (b) shows that they have the similar performance. Only when $\tau < 200$, KLS has a slight advantage over DCCplex and Nukplex in terms of run time. This is because all the algorithms employ the reduction procedure that can reduce most of the vertices in a given large sparse graph. Specifically, for 87% large sparse graphs, all the heuristic algorithms obtain the best solution (Line 28 in Algorithm 2) within 10 seconds.

Comparisons against MKPP exact algorithms. The two exact algorithms do not perform well on all the classic instances, as neither of them can obtain the same solution as Nukplex for at least 70% of these instances. However, the two exact algorithms exhibit the excellent performance on the large sparse instances, with KpLeX-Gap failing to obtain only 18 best solutions. Moreover, Nukplex outperforms DiseMKP and KpLeX-Gap for 18 and 30 instances by running only one seed (e.g., seed=1), while it is defeated for only 0 and 1 instance (*soc-orkut* with k=5), respectively. It is worth noting that Nukplex successfully finds the best solution for this instance by utilizing different seed values ranging from 1 to 10. Once again, the comparison with the two exact algorithms highlights the great performance of Nukplex. We provide some additional comparisons with the exact algorithms in the supplementary material.

7.2 The Effectiveness of the Proposed Strategies

In this subsection, we present the effectiveness of the proposed strategies in Table 3. To confirm the effectiveness of our proposed RBCC strategy, we compare Nukplex with three alternative algorithms where Nukp1 uses BoundedCC instead of RBCC, Nukp2 adopts DCC instead of RBCC, and Nukp3 applies SCC [Wang *et al.*, 2020] instead of RBCC. The results intuitively show that RBCC clearly improves the performance of the MKPP. In addition, we compare Nukplex with one alternative algorithm Nukp4 that ignores a core-based perturbation strategy in Nukplex. The results demonstrate that our proposed method plays a key role in the Nukplex.

Three modified versions of Nukplex are also proposed to verify the effectiveness of the proposed vertex selection strategy, especially on classic graphs. We compared alternative with three alternative versions: (1) Nukp5 uses the random-based method [Zhou and Hao, 2017; Chen *et al.*, 2020] in $ConstrAddSet$ instead of our proposed add rule; (2) Nukp6 utilizes the random-based method in $ConstrSwapSet$ instead of our proposed swap rule; (3) Nukp7 replaces our add rule with the structure-based method [Pullan, 2021]. As shown in Table 3, both the proposed add and swap rules perform better than previous vertex selection rules. Especially for the proposed swap rule, it clearly improves the performance of the Nukplex.

8 Conclusion

This paper proposes a local search algorithm called Nukplex for the MKPP. It mainly comprises a relaxed bounded configuration checking strategy, a novel solution information-based vertex selection strategy and a core-based perturbation strategy. Results show that Nukplex significantly outperforms the state-of-the-art heuristic algorithms for the MKPP.

In the future, we intend to conduct further research on variants of CC and core-based perturbation strategies for other cohesive subgroups, as well as explore additional vertex properties to enhance the performance of our proposed algorithm.

Acknowledgements

This work is supported by NSFC under Grant No. (61806050, 61976050), CCF Huawei Hu Yanglin Foundation Theoretical Computer Science Project CCF-HuaweiLK2023001, the Fundamental Research Funds for the Central Universities 2412022ZD015, 2412023YQ003, Jilin Science and Technology Department (YDZJ202201ZYTS412, 20240602005RC), Ministry of Education 2021BCF01002.

References

[Balasundaram *et al.*, 2011] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.

[Balasundaram, 2007] Balabhaskar Balasundaram. *Graph theoretic generalizations of clique: Optimization and extensions*. Texas A&M University, 2007.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10):1672–1696, 2011.

[Chang *et al.*, 2022] Lijun Chang, Mouyi Xu, and Darren Strash. Efficient maximum k-plex computation over large sparse graphs. *Proceedings of the VLDB Endowment*, 16(2):127–139, 2022.

[Chen *et al.*, 2020] Peilin Chen, Hai Wan, Shaowei Cai, Jia Li, and Haicheng Chen. Local search with dynamic-threshold configuration checking and incremental neighborhood updating for maximum k-plex problem. In *AAAI*, pages 2343–2350, 2020.

[Chen *et al.*, 2021] Jiejiang Chen, Shaowei Cai, Shiwei Pan, Yiyuan Wang, Qingwei Lin, Mengyu Zhao, and Minghao Yin. Nuqclq: An effective local search algorithm for maximum quasi-clique problem. In *AAAI*, pages 12258–12266, 2021.

[Chen *et al.*, 2022] Jiejiang Chen, Yiyuan Wang, Shaowei Cai, Minghao Yin, Yupeng Zhou, and Jieyu Wu. Nukcp: An improved local search algorithm for maximum k-club problem. In *AAAI*, pages 10146–10155, 2022.

[Chen *et al.*, 2023] Jiejiang Chen, Shaowei Cai, Yiyuan Wang, Wenhao Xu, Jia Ji, and Minghao Yin. Improved local search for the minimum weight dominating set problem in massive graphs by using a deep optimization mechanism. *Artificial Intelligence*, 314:103819, 2023.

[Conte *et al.*, 2018] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2k: scalable community detection in massive networks via small-diameter k-plexes. In *SIGKDD*, pages 1272–1281, 2018.

[Dolan and Moré, 2002] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[Gao *et al.*, 2018] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An exact algorithm for maximum k-plexes in massive graphs. In *IJCAI*, pages 1449–1455, 2018.

[Glover, 1989] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[Gschwind *et al.*, 2018] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 234:131–138, 2018.

[Gujjula *et al.*, 2014] Krishna Reddy Gujjula, Krishnan Ayalur Seshadrinathan, and Amirhossein Meisami. A hybrid metaheuristic for the maximum k-plex problem. In *Examining Robustness and Vulnerability of Networked Systems*, pages 83–92. IOS Press, 2014.

[Jiang *et al.*, 2021] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A new upper bound based on vertex partitioning for the maximum k-plex problem. In *IJCAI*, pages 1689–1696, 2021.

[Jiang *et al.*, 2023] Hua Jiang, Fusheng Xu, Zhifei Zheng, Bowen Wang, and Wei Zhou. A refined upper bound and inprocessing for the maximum k-plex problem. In *IJCAI*, pages 5613–5621, 2023.

[Jin *et al.*, 2022] Yan Jin, John H Drake, Kun He, and Una Benlic. Reinforcement learning based coarse-to-fine search for the maximum k-plex problem. *Applied Soft Computing*, 131:109758, 2022.

[Johnson and Trick, 1996] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[Kondo *et al.*, 2012] Keisuke Kondo, Toshihiro Okubo, et al. Structural estimation and interregional labour migration: Evidence from japan. Technical report, Keio/Kyoto Joint Global COE Program, 2012.

[López-Ibáñez *et al.*, 2016] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[McClosky *et al.*, 2012] Benjamin McClosky, Illya V Hicks, et al. Combinatorial algorithms for the maximum k-plex problem. *Journal of combinatorial optimization*, 23(1):29, 2012.

[Miao and Balasundaram, 2017] Zhuqi Miao and Balabhaskar Balasundaram. Approaches for finding cohesive subgroups in large-scale social networks via maximum k-plex detection. *Networks*, 69(4):388–407, 2017.

[Moser *et al.*, 2012] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *Journal of combinatorial optimization*, 24:347–373, 2012.

[Pattillo *et al.*, 2013] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

[Pullan, 2021] Wayne Pullan. Local search for the maximum k-plex problem. *Journal of Heuristics*, 27(3):303–324, 2021.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, pages 4292–4293, 2015.

[Seidman and Foster, 1978] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[Shahinpour and Butenko, 2013] Shahram Shahinpour and Sergiy Butenko. Algorithms for the maximum k-club problem in graphs. *Journal of Combinatorial Optimization*, 26(3):520–554, 2013.

[Shirokikh, 2013] Oleg A Shirokikh. *Degree-based clique relaxations: theoretical bounds, computational issues, and applications*. PhD thesis, University of Florida, 2013.

[Trukhanov *et al.*, 2013] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56:113–130, 2013.

[Wang *et al.*, 2020] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. Sccwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.

[Wang *et al.*, 2023] Zhengren Wang, Yi Zhou, Chunyu Luo, and Mingyu Xiao. A fast maximum k-plex algorithm parameterized by the degeneracy gap. In *IJCAI*, pages 5648–5656, 2023.

[Wasserman and Faust, 1994] Stanley Wasserman and Katherine Faust. Social network analysis: Methods and applications. 1994.

[Wu and Hao, 2015] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[Wu *et al.*, 2019] Kuixian Wu, Jian Gao, Rong Chen, and Xianji Cui. Vertex selection heuristics in branch-and-bound algorithms for the maximum k-plex problem. *International Journal on Artificial Intelligence Tools*, 28(05):1950015, 2019.

[Xiao *et al.*, 2017] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. A fast algorithm to compute maximum k-plexes in social network analysis. In *AAAI*, pages 919–925, 2017.

[Xu *et al.*, 2007] Ke Xu, Frederic Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8-9):514–534, 2007.

[Zhou and Hao, 2017] Yi Zhou and Jin-Kao Hao. Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research*, 86:65–78, 2017.

[Zhou *et al.*, 2021] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *AAAI*, pages 12453–12460, 2021.

[Zhu *et al.*, 2020] Jinrong Zhu, Bilian Chen, and Yifeng Zeng. Community detection based on modularity and k-plexes. *Information Sciences*, 513:127–142, 2020.