NukCP: An Improved Local Search Algorithm for Maximum k-Club Problem

Jiejiang Chen[1], Yiyuan Wang[1,3*], Shaowei Cai[2,4], Minghao Yin[1,3*], Yupeng Zhou[1,3*], Jieyu Wu[1]

[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[3]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
[4]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
chenjj016@nenu.edu.cn, yiyuanwangjlu@126.com, caisw@ios.ac.cn, {ymh, zhouyp605}@nenu.edu.cn, wjy4204@163.com

Abstract

The maximum k-club problem (MkCP) is an important clique relaxation problem with wide applications. Previous MkCP algorithms only work on small-scale instances and are not applicable for large-scale instances. For solving instances with different scales, this paper develops an efficient local search algorithm named NukCP for the MkCP which mainly includes two novel ideas. First, we propose a dynamic reduction strategy, which makes a good balance between the time efficiency and the precision effectiveness of the upper bound calculation. Second, a stratified threshold configuration checking strategy is designed by giving different priorities for the neighborhood in the different levels. Experiments on a broad range of different scale instances show that NukCP significantly outperforms the state-of-the-art MkCP algorithms on most instances in terms of solution quality.

Introduction

A clique is a subset of vertices of an undirected graph in which each pair of vertices are adjacent. Clique is one of the basic concepts of graph theory and has been widely studied (Ouyang et al. 1997; Butenko and Wilhelm 2006). However, the constraint condition of clique model is too strict for many real-world applications since the aim of these applications is to find some dense structures rather than a complete subgraph. These structures can usually be seen as clique relaxation models and are mainly divided into two categories: density-based models such as k-plex (Gao et al. 2018) and quasi-clique (Chen et al. 2021) as well as diameter-based models such as k-clique (Cavique, Mendes, and Santos 2009) and k-club (Shahinpour and Butenko 2013b).

In this paper, we focus on studying the maximum k-club problem (MkCP) which has been used in various domains. For example, the MkCP can help to facilitate the search on the internet since it can cluster topically related information (Pattillo, Youssef, and Butenko 2013). In social network, the well-known "a friend of a friend" concept can be modeled as k-club and the problem of finding a low-diameter community can be encoded to the MkCP (Goodreau, Kitts, and Morris 2009). The MkCP is also used to analyze the roles of functional modules by mining important substructures in biological network (Balasundaram, Butenko, and Trukhanov 2005; Jia et al. 2018).

Given a graph $G = (V, E)$ and a fixed integer k, a k-club S is a subset of vertices inducing a subgraph of diameter at most k. It is easy to see that 1-club is a clique. The MkCP aims to identify the k-club with the maximum size in a graph. It is NP-hard, even for any fixed $k > 1$ (Bourjolly, Laporte, and Pesant 2002). Up to now, there are mainly two types of algorithms for the MkCP, i.e., exact algorithms and heuristic algorithms.

Several exact algorithms have been designed to solve the MkCP. Bourjolly et al. (2002) proposed a classic branch-and-bound algorithm for the MkCP and could solve the instances involving up to 200 vertices. The branch-and-bound algorithms for the MkCP were further improved by considering the k-coloring number as an upper bound (Pajouh and Balasundaram 2012) and designing a dynamic data structure (Chang et al. 2013). Recently, another paradigm solved the MkCP by using different integer linear programming formulations (Almeida and Carvalho 2012, 2014a; Veremyev et al. 2021). Although exact algorithms can guarantee the optimality of their solutions, they may fail to solve the problem within acceptable time, especially for large-scale instances.

In practice, for solving the large-scale MkCP instances, researchers resort to designing heuristic algorithms for obtaining good solutions. Bourjolly et al. (2000) proposed three simple and effective heuristic algorithms for the MkCP, including Constellation, Drop and k-Clique & Drop. Shahinpour and Butenko (2013a) developed a variable neighborhood search called VNS for the MkCP and firstly tested the performance of VNS on the DIMACS benchmark. Afterwards, two hybrid algorithms including mS_IP specialized for the 2-club problem and mB_IP specialized for the 3-club problem were introduced by combining heuristic algorithms and integer linear programming formulations (Almeida and Carvalho 2014b). In the same work, Almeida and Carvalho (2014b) also proposed a heuristic algorithm called Backbone and proved its superiority over the previous algorithm Constellation on the 3-club problem. Moradi and Balasundara (2018) proposed a heuristic algorithm for the MkCP called ITDBC which used a combination of graph decomposition and model decomposition techniques. In the ITDBC, a reduction method called TRIM played an impor-

[*]Corresponding author

tant role in cutting down the search space. Results showed that ITDBC achieved the best results on most instances.

In this paper, we develop an efficient local search algorithm named NukCP which can apply to different scale instances and performs better on almost all benchmarks. There are two main novel ideas in our algorithm.

The first idea is called dynamic reduction method (DRM) designed for quickly reducing the size of graphs. The previous reduction method TRIM (2018) used a kind of static upper bound calculation, which is time-consuming and thus is not applicable for large-scale instances. Compared to TRIM, the DRM allows to delete vertices during the calculation of upper bounds. Additionally, the DRM introduces a novel method to dynamically update the upper bound values instead of recalculating them as the TRIM does. These two techniques make our reduction method much faster than TRIM. Experiments show that the DRM can achieve good practical results within short time.

The second idea is an improved version of the configuration checking (CC) strategy called stratified threshold configuration checking (STCC). CC firstly proposed by Cai et al. (2011) was used to overcome the cycling problem during local search. Recently, different variants of CC have been designed for solving the clique related problems such as SCC for the maximum weight clique problem (Wang, Cai, and Yin 2016), DCC for the maximum k-plex problem (Chen et al. 2020) and BoundedCC for the maximum quasi-clique problem (Chen et al. 2021). Different from the above problems, the MkCP needs to consider the multi-level neighborhood of vertices due to its characteristics. In our work, we take the characteristics into account and maintain the multi-level neighborhood information. Although previous CC version named CC^{2}V3 (Wang et al. 2018) considers the two-level neighborhood, for the MkCP we need to take more than two-level neighborhood into account, which leads to a serious problem, i.e., how to maintain the neighborhood information. To address this, the spanning tree is used to dynamically maintain the multi-level neighborhood of vertices in the candidate solution.

We carry out extensive experiments to evaluate the NukCP on the benchmarks used in the literature for the MkCP as well as a suite of massive graphs (Rossi and Ahmed 2015). Compared with three state-of-the-art heuristic algorithms, NukCP obtains the best results for almost all benchmarks. Besides, our experimental analyses verify the effectiveness of the proposed strategies.

The remainder of the paper is organized as follows. The next section introduces some basic definitions. Section 3 presents a new reduction strategy. Section 4 presents a variant of CC designed for the MkCP. Section 5 describes our NukCP algorithm. Experimental results are shown in Section 6 and Section 7 gives concluding remarks.

Preliminaries

An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set E. Two vertices are neighbors if they belong to one edge. We denote $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ as the set of neighbors of a vertex v and its degree is $deg_G(v) = |N_G(v)|$. Given a pair of vertices $u, v \in V$, the distance $dist_G(u, v)$ is the number of edges in a shortest path connecting them and $dist_G(v, v) = 0$ particularly. The diameter of G denoted as $diam(G)$ is the maximum distance between any pair of vertices. We define the i-th level neighborhood of v in G as $N_{i,G}(v) = \{u \in V \mid dist_G(u, v) = i\}$. $N_{i,G}[v] = N_{i,G}(v) \cup \{v\}$ and $N^k_G[v] = \bigcup_{i=1}^{k} N_{i,G}[v]$. Thus, $N_{1,G}(v) = N_G(v)$. For $S \subseteq V$, $G[S] = (V_S, E_S)$ is a subgraph in G induced by S whose vertex set is S and whose edge set consists of all of the edges in E that have both endpoints in S.

Given a graph G and a fixed integer k, a k-club S is a subset of V such that $diam(G[S]) \leq k$. The maximum k-club problem (MkCP) is to find a k-club with the most vertices.

A Dynamic Reduction Strategy

Although reduction methods have been widely used in the clique related problems (Cai and Lin 2016; Jiang, Li, and Manyà 2016; Wang et al. 2020a; Zhou et al. 2021), we are aware of only one reduction method for the MkCP, which is named TRIM (Moradi and Balasundaram 2018). In this section, we propose a novel reduction method for the MkCP, which is built upon the rules of TRIM but significantly faster than TRIM.

Previous Static Reduction Strategy

For the MkCP, the general principle for reduction is as follows. For a graph G, an upper bound function calculates for each vertex a value $ub_G(v)$ such that the size of any k-club that v belongs to is at most $ub_G(v)$. A lower bound function calculates a value $lb(G)$ such that the size of the largest k-club in G is not smaller than $lb(G)$. All vertices with $ub_G(v) \leq lb(G)$, along with their incident edges, can be safely deleted since they cannot be part of any optimal solution.

The TRIM reduction method employs simple but effective lower bound and upper bound.

- $lb(G) = max\{|N_G^{\lfloor k/2 \rfloor}[v]| \mid v \in V\}$
- $ub_G(v) = |N^k_G[v]|$

When applying reduction rules, the graph G always refers to the current graph in process and thus can be omitted. For convenience, $lb(G)$ and $ub_G(v)$ are denoted as lb and $ub(v)$. TRIM works in an iterative way and each iteration is consisted of two steps as below.

- Step 1. Calculate $X = \{v \in V \mid ub(v) \leq lb\}$;
- Step 2. If X is not empty, then $V = V \setminus X$ and TRIM goes to step 1. Otherwise, TRIM breaks the iteration.

In practice, TRIM is time-consuming and not suitable for the massive graphs due to the following two reasons. (1) The deletion of a vertex v would change the ub values of those vertices in $N^k_G[v]$. TRIM recalculates ub values for such vertices. What is worse, the ub values for some vertices are recalculated multiple times. (2) In each iteration, TRIM first calculates ub values for all vertices, and then removes those vertices satisfying the reduction condition. In this way, TRIM does not update the ub values in real time based on the current simplified graph. This leads to more iterations than necessary for the reduction process to converge.

Dynamic Reduction Method

In order to address the drawback of TRIM, we develop a new reduction method for the MkCP called dynamic reduction method (DRM). There are two main ideas in the DRM, with the aim of resolving the two issues of TRIM. Firstly, DRM only calculates the ub values once for all vertices and then dynamically updates the ub values instead of recalculating them whenever a vertex is deleted. Secondly, DRM allows a vertex to be deleted during the process of calculating its ub value — this strategy not only decreases the iterations of reduction, but also accelerates the subsequent calculation or updating of ub values as the computation is executed on the simplified graph.

The proposed DRM is consisted of two phases: the dynamic calculation phase and the iterative deletion phase. During the DRM, the calculation of ub is only executed once at the beginning of the algorithm and the iterative deletion phase is used for further reduction. Before presenting our DRM, we first introduce a novel dynamic maintenance rule named dynamic update rule (DUR) for updating the neighborhood of a deleted vertex.

Dynamic Update Rule. When a vertex $v \in V$ is deleted from G, $ub(u) = ub(u) - 1$ for $\forall u \in N_G^k[v]$.

The above rule has a low time complexity of $O(|V|)$. Based on the DUR, we present the dynamic calculation phase whose implementation efficiency is closely related to the sequence of vertices and the iterative deletion phase which is implemented by an auxiliary queue.

Dynamic Calculation Phase. Initially, the positions of vertices are arranged in an ascending order of their degrees, $V = \{v_1, v_2, \ldots, v_n\}$, s.t. $deg_G(v_1) \leq deg_G(v_2) \leq \cdots \leq deg_G(v_n)$. If $ub(v_i) \leq lb$, then v_i is deleted from G in advance, and at the same time the ub values for its neighborhood $v_j \in N_G^k[v_i]$ with $j < i$ are updated by the DUR.

Iterative Deletion Phase. An auxiliary queue is used to store all vertices $v \in V$ with $ub(v) \leq lb$. In each iteration, a vertex in the queue is deleted from G and the ub values of each vertex $u \in N_G^k[v]$ are updated by the DUR. If $ub(u) \leq lb$ and u is not in the queue, then u is added into the queue. The loop continues until the queue is empty.

Notice that during the first phase, when deleting v_i from G, we do not update the ub value of the vertex v_j whose position is behind v_i (i.e., $j > i$) since $ub(v_j)$ has not been initialized yet. Additionally, the deletion of v_i reduces the size of G, which makes the subsequent process of calculating ub values faster. Thus, before calculating ub values, we sort all vertices in an ascending order according to their degrees, trying to delete vertices as early as possible, which accelerates the overall deletion process.

As explained above, the DUR avoids the recalculation of ub values. On the other hand, this comes with a price that the ub values calculated in this way may be actually larger than the ub of TRIM whose value is always equal to $|N_G^k|$. It is mainly caused by the following situation: when a vertex v is deleted from G, the $|N_G^k[u]|$ values for $u \in N_G^k[v]$ may decrease by more than one while the DUR just subtracts one for $ub(u)$.

We use an example to illustrate this situation in Figure 1. Assume that an original graph is $G = (V, E)$ and

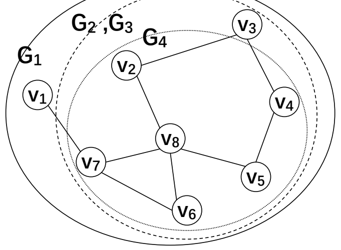

Figure 1: An example of the DRM for the 2-club problem.

$V = \{v_1, \ldots, v_8\}$ whose positions have been already arranged. In order to show the changes in the size of the graph during the first phase, we use $G_1, G_2, \ldots, G_8$ to denote the corresponding graphs where $V_{i+1} = V_i \setminus \{v_i\}$ if v_i is deleted and $V_{i+1} = V_i$ otherwise. We can easily get $lb = |N_G^1[v_8]| = 5$. At the beginning, there are no vertices being deleted inducing that $G_1 = G$. Based on the G_1, $ub_{G_1}(v_1)$ is calculated. Because $ub_{G_1}(v_1) = 4 < lb$, v_1 should be deleted and $V_2 = V_1 \setminus \{v_1\}$. Afterwards, we cannot delete v_2 since $ub_{G_2}(v_2) = 7$, and thus $G_3 = G_2$. $ub_{G_3}(v_3) = 5$ induces that v_3 should be deleted. When deleting v_3, we update its neighborhood according to the DUR which decreases $ub(v_2)$ by one, i.e., $ub_{G_3}(v_2) = 6$. However, if we recalculate $ub(v_2)$ on G_3 as what TRIM does, the value of $|N_{G_3}^2[v_2]|$ should be 5. This is because the deletion of v_3 makes v_4 no longer be in the neighborhood of v_2. Thus, in some cases, our upper bound value is larger than the one calculated by TRIM.

Even though our method may sacrifice the precision of the ub values, it has greatly improved the efficiency of updating process and can be used for massive graphs. In fact, our experimental results show that the reduction ability of our dynamic reduction method is close to that of TRIM, whilst it greatly improves the efficiency.

The DRM Function The pseudo code of DRM is presented in Algorithm 1, which includes the dynamic calculation phase (lines 2–13) and the iterative deletion phase (lines 14–18). Input parameter $mode$ = 1 means the algorithm calls the DRM function on the initialization phase, while $mode$ = 2 means that the DRM function is used when the algorithm obtains a better solution during the local search phase. At first, the algorithm initializes a deletion queue named Q_d as an empty set (line 1). When entering the first phase, the algorithm first sorts all vertices in an ascending order according to their degrees (line 3). After that, for each vertex v_i, its $ub(v_i)$ value will be calculated. If $ub(v_i) \leq lb$, then v_i will be deleted from V in advance (line 7). For v_j in the v_i's neighborhood and $j < i$, $ub(v_j)$ will be updated according to the DUR (lines 8–9). If $ub(v_j) \leq lb$, then v_j will be added into Q_d (line 10). If NukCP calls the DRM during the local search phase (i.e., $mode$ = 2), then the vertices $v_i \in V$ with $ub(v_i) \leq lb$ will be added into Q_d (lines 11–13).

In the iterative deletion phase, during each iteration (lines 14–18), a vertex $v \in Q_d$ will be deleted from G and its neighborhood should be updated according to the DUR. Af-

Algorithm 1: the DRM function

```
Input: graph $G = (V, E)$, lower bound $lb$, parameter $k$,
       reduction mode $mode$
Output: the graph after reduction $G$
1  $Q_d := \emptyset$;
2  if $mode == 1$ then
3      sort $v \in V$ in an ascending order based on $deg_G$;
4      for $i := 1; i \le |V|; i := i + 1$ do
5          $ub(v_i) := |N_G^k[v_i]|$;
6          if $ub(v_i) \le lb$ then
7              $V := V \setminus \{v_i\}$;
8              foreach $v_j \in N_G^k[v_i]$ && $j < i$ do
9                  $ub(v_j) := ub(v_j) - 1$;
10                 if $ub(v_j) \le lb$ then $Q_d := Q_d \cup \{v_j\}$ ;
11 else if $mode == 2$ then
12     foreach $v_i \in V$ do
13         if $ub(v_i) \le lb$ then $Q_d := Q_d \cup \{v_i\}$ ;
14 while $Q_d$ is not empty do
15     pop a vertex $v$ in $Q_d$ and $V := V \setminus \{v\}$;
16     foreach $u \in N_G^k[v] \setminus \{v\}$ do
17         $ub(u) := ub(u) - 1$;
18         if $ub(u) \le lb$ then $Q_d := Q_d \cup \{u\}$ ;
19 return $G$;
```

ter updating the upper bound of vertex u, the algorithm adds it into Q_d if $ub(u) \le lb$ (line 18). Finally, the graph after reduction is returned (line 19).

Stratified Threshold Configuration Checking

Configuration checking (CC) was firstly proposed to deal with the cycling problem in local search (Cai, Su, and Sattar 2011) and has been successfully used in several NP-hard problems. The CC is mainly based on the definition of *configuration* of the vertex which refers to the states (i.e., in solution or not) of its neighbors. For $u \in N_{1,G}(v)$, once u changes its state, then we say that the *configuration* of v has been changed. Only the vertices whose *configuration* have changed are allowed to add back into the candidate solution.

Intuition of STCC

Recently, many variants of CC for clique relaxation problems have been designed, such as DCC for the maximum k-plex problem (Chen et al. 2020) and BoundedCC for the maximum quasi-clique problem (Chen et al. 2021). The proposed DCC and BoundedCC strategies adopted the first level neighborhood (i.e., $N_{1,G}$) as the configuration of vertex. Different from the above problems, the relaxation constraint of the MkCP considers the distance between the vertices, which intuitively refers to the multi-level neighborhood of the corresponding vertices. Thus, it is not applicable to directly use the previous CC strategies into solving the MkCP.

Based on the above considerations, we propose a stratified version of DCC called the stratified threshold configuration checking (STCC) strategy for the MkCP to distinguish the effects caused by the states changing of different neighborhood. The reason for choosing DCC as the basic strategy for expansion is that the search space for clique relaxation problems is usually relatively concentrated, which will result in that the high-degree vertices are very likely to change their configurations. The introduction of *threshold* makes those vertices that frequently change their states have more restrictions, thereby increasing their forbidding strength.

Intuitions underlying the STCC strategy are given below. When adding a vertex into the candidate solution, it is quite reasonable to allow the multi-level neighborhood of the added vertex to be added by giving them different priorities. On the other hand, the removal operation (i.e., removing a vertex from the candidate solution) can hardly improve the quality of solution. In this case, we keep the previous priorities for the multi-level neighborhood of the removal vertex.

Data Structure of STCC

Different from DCC and BoundedCC which only preserve the first level neighborhood, we need to maintain the first k-level neighborhood of each vertex in the candidate solution S as the configuration information. Moreover, the first k-level neighborhood of vertices for the MkCP will be dynamically changed with respect to S.

In order to facilitate the dynamic maintenance of the neighborhood information, for each $v \in S$, we build a spanning tree for it, denoted as $T^v = (V_T^v, E_T^v)$, where $V_T^v = \{u \in V \mid dist_{G[S\cup\{u\}]}(u, v) \le k\}$. $dist_{T^v}(u, v)$ is used to denote the depth value of vertex u in the T^v. During the search process, we maintain $|S|$ spanning trees. Three corresponding updating rules of spanning tree are described as follows.

Constructing Rule. For $v \in S$, a spanning tree $T^v = (V_T^v, E_T^v)$ is constructed via using breadth-first search. During this process, three expansion ways are used. 1) For $u_1 \notin S$ and $dist_{G[S\cup\{u_1\}]}(u_1, v) \le k$, we do not expand it and directly mark it as a leaf vertex. 2) For $u_2 \in S$ and $dist_{G[S]}(u_2, v) < k$, u_2 will be expanded. If u_2 cannot expand any vertices, then we mark u_2 as a leaf vertex. 3) For $u_3 \in S$ and $dist_{G[S]}(u_3, v) = k$, u_3 needs to be marked as a leaf vertex.

Adding Rule. When vertex v is added into S, T^v will be generated according to the constructing rule. For $\forall u \in (T^v \cap S)$, T^u needs to be updated in the following method. For v and $\forall w \in V_T^u$ with $dist_{T^u}(u, w) > dist_{T^u}(u, v)$, these vertices should be re-extended by using the expansion ways of the constructing rule.

Removing Rule. When vertex v is removed from S, we delete its spanning tree T^v and update T^u for $\forall u \in (T^v \cap S)$. For $\forall w \in V_T^u$ with $dist_{T^u}(u, w) \ge dist_{T^u}(u, v)$, these vertices should be re-extended by using the expansion ways of the constructing rule.

The vertices inside S constitute the trunk or leaf vertices of the spanning tree, and some vertices outside S constitute the remaining leaf vertices. The complexity of constructing or updating a spanning tree T^v is $O(|V_{N_G^k[v]}| + |E_{N_G^k[v]}|)$. To make the updating rules more comprehensive, we show an example in Figure 2 with regard to maintaining T^{v_1} when adding v_3 and then removing v_5.

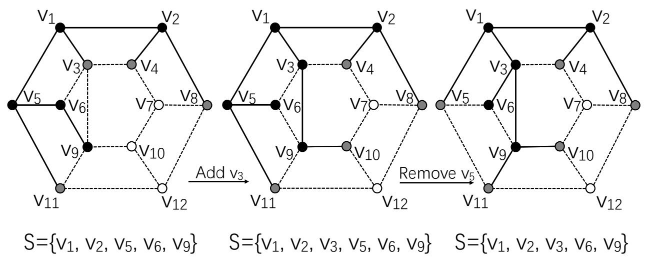

Figure 2: An example of updating spanning tree for the 3-club problem where the black vertices denote the vertices inside S, the grey vertices denote the vertices in T^{v_1} but not inside S and the solid edges denote the edges in T^{v_1}.

Update Rule for STCC

For $\forall v \in V$, the proposed STCC strategy is implemented with two arrays $conf[v]$ and $thred[v]$ to denote the configuration and threshold of vertices, respectively. Only when $conf[v] \geq thred[v]$, v is allowed to be added into S. The novel STCC strategy is specified by the following rules.

STCC-InitialRule. For $\forall v \in V$, $conf[v]$ and $thred[v]$ are both initialized to k.

STCC-AddRule. When v is added into the candidate solution, for $\forall u \in V_T^v$, $conf[u]$ is increased by $k + 1 - dist_{T^v}(u, v)$ and $thred[v]$ is increased by 1.

STCC-RemoveRule. When v is removed from the candidate solution, $conf[v]$ is reset to 0.

In the above rules, we increase $conf[u]$ by different values according to their $dist_{T^v}$ to reflect the impact of changes in the states of neighborhood at different levels.

The NukCP Algorithm

According to the above strategies, we propose a local search algorithm for the MkCP named NukCP. Before introducing the NukCP, we present the scoring function used in our work, which has been used in (Bourjolly, Laporte, and Pesant 2000; Almeida and Carvalho 2014b) for the MkCP. We denote our scoring function as $score(v)$ for $\forall v \in V$.

$$score(v) = |\{u \in S \mid dist_{G[S \cup \{v\}]}(u, v) \leq k\}|$$

Based on the above scoring function, our vertex selection rules are given as below.

Selection Adding Rule. Select a vertex $v \in V \setminus S$ with the highest $score(v)$ value, breaking ties by the oldest one[1].

Selection Removing Rule. Select a vertex $v \in S$ with the lowest $score(v)$ value, breaking ties by the oldest one.

Note that only when the $score$ values of all vertices in S are equal to $|S|$, then we say S is a feasible solution for the MkCP. In addition, for $k = 3$, we use a tighter lower bound $lb = max\{|N_{1,G}[u] \cup N_{1,G}[v]| \mid \{u, v\} \in E\}$ proposed by Almeida and Carvalho (2014b).

[1]The age of a vertex is the number of steps since its state was last changed.

Algorithm 2: the NukCP algorithm

```
Input: graph $G$, the cutoff time, parameter $k$
Output: the best $k$-club $S^*$ found
1  $S^* := \emptyset$;
2  if $k \neq 3$ then $lb := max\{|N_G^{\lfloor k/2 \rfloor}[v]| \mid v \in V\}$;
3  else $lb := max\{|N_{1,G}[u] \cup N_{1,G}[v]| \mid \{u, v\} \in E\}$;
4  $G := DRM(G, lb, k, 1)$;
5  while elapsed time $<$ cutoff do
6      $S := InitConstruct(G)$;
7      $S_{lbest} := ClubSearch(G, S)$;
8      if $|S_{lbest}| > |S^*|$ then
9          $S^* := S_{lbest}$;
10         $G := DRM(G, |S^*|, k, 2)$;
11         if $|V_G| \leq |S^*|$ then return $S^*$;
12 return $S^*$;
```

The Main Framework of NukCP

The pseudo code of NukCP is outlined in Algorithm 2. At first, the algorithm initializes S^* (line 1) and calculates the lower bound lb according to k (lines 2-3). The algorithm reduces the original graph by calling the DRM (line 4).

In each loop (lines 5–11), the algorithm first constructs an initial candidate solution S by calling our $InitConstruct$ process (line 6). Specifically, $InitConstruct$ first selects a random vertex $v \in V$, and then sets $S = N_G^k[v]$ as an initial solution. $InitConstruct$ iteratively removes vertices according to the selection removing rule until S becomes a feasible solution, and S will be returned. Afterwards, the algorithm calls the $ClubSearch$ process to improve the current solution (line 7). If the local best solution S_{lbest} in this search trajectory is better than S^*, then S^* is updated by S_{lbest} and the algorithm tries to reduce the graph G again (lines 8–10). If the size of remaining vertices in G is smaller than $|S^*|$ which means the optimal solution is found, then we can return S^* in advance (line 11). Otherwise, we return S^* when the time limit is reached (line 12).

The $ClubSearch$ Function

After getting the initial solution, the algorithm calls the $ClubSearch$ function to improve this solution. We formalize the $ClubSearch$ function in Algorithm 3 as below. At the beginning, $step$ and S_{lbest} are set to 0 and S, respectively (line 1). The spanning tree of each vertex in S is initialized according to the constructing rule (line 2). The algorithm sets the $conf$ and $thred$ of each vertex to k (line 3). After then, the algorithm searches for a local optimal solution denoted by S_{lbest} (lines 4–18). Finally, the algorithm returns S_{lbest} when $step$ reaches parameter $stepMax$ (line 19).

During each step, if S is a feasible solution, S_{lbest} is updated by S and $step$ is reset to 0 (line 6). The algorithm selects a vertex by using the selection adding rule and the STCC strategy, and then adds it into S (line 7). Otherwise, if S is not a feasible solution, the algorithm uses the above same adding strategy to add at most two vertices into S (lines 9-11). This is because for the MkCP, adding two vertices may make the solution become feasible while adding only one vertex fails to obtain a feasible solution in some

Algorithm 3: ClubSearch(G,S)

```
Input: Graph $G$, a feasible solution $S$
Output: the best local solution $S_{lbest}$ found
1  $step := 0$, $S_{lbest} := S$;
2  build spanning tree $T^w$ for $\forall w \in S$ according to
     Constructing Rule;
3  $conf[v] := thred[v] := k$ for $\forall v \in V$;
4  while $step < stepMax$ do
5      if $S$ is a k-club then
6          $S_{lbest} := S$, $step := 0$;
7          select $v_1$ with $conf[v_1] \geq thred[v_1]$ according to
             Selection Adding Rule;
8          $S := S \cup \{v_1\}$ and update the spanning trees,
             $conf$ and $thred$;
9      while $S$ is infeasible and $|S| \leq |S_{lbest}| + 2$ do
10         select $v_2$ with $conf[v_2] \geq thred[v_2]$ according to
             Selection Adding Rule;
11         $S := S \cup \{v_2\}$ and update the spanning trees,
             $conf$ and $thred$;
12     if $S$ is infeasible then
13         select $u_1$ according to Selection Removing Rule;
14         $S := S \setminus \{u_1\}$ and update the spanning trees and
             $conf$;
15     if $S$ is infeasible and with probability $\alpha$ then
16         select $u_2$ according to Selection Removing Rule;
17         $S := S \setminus \{u_2\}$ and update the spanning trees and
             $conf$;
18     $step := step + 1$;
19 return $S_{lbest}$;
```

cases, which has also been discussed in the previous literature (Shahinpour and Butenko 2013a). After that, the algorithm tries to remove a vertex according to the selection removing rule (lines 12–14). If S is still infeasible, another removed vertex is selected with the probability α (lines 15–17). During the whole step, after a selected vertex has been operated, the corresponding spanning tree, $conf$ and $thred$ should be updated accordingly.

Experimental Evaluation

We carry out experiments to evaluate NukCP on a broad range of random and DIMACS benchmarks as well as massive graphs for k = 2,3,4. We compare NukCP with three previous algorithms, including VNS (2013a), mS/B (2014b)[2] and ITDBC (2018). Note that mS/B is only designed for k = 2 and 3.

All algorithms were implemented in C++ and compiled by g++ with '-O3' option. CPLEX 12.63[3] and Gurobi [4] are used in the mS/B and ITDBC, respectively. We set the same parameters as what described in the corresponding literature and optimize these parameters for the new added instances. The parameters $stepMax$ and α in NukCP are set to 100 and 0.6 according to our preliminary experiment. All experiments are run on Intel Xeon E5-2640 v4 @ 2.40GHz CPU with 128GB RAM under CentOS 7.5.

Instance	k	NukCP	VNS	ITDBC	mS/B
Family		$\overline{max}$	$\overline{max}$	$\overline{max}$	$\overline{max}$
n100d0.03	3	**16.4**	**16.4**	**16.4**	16.3
n100d0.04	3	**24.2**	**24.2**	**24.2**	23.4
n100d0.02	4	**20.9**	**20.9**	20.8	
n100d0.03	4	**35.3**	35.2	35.2	
n100d0.04	4	**58.7**	58.5	58.6	
n200d0.02	4	**55.2**	54.9	54.9	
n300d0.015	4	**62.7**	61.4	**62.7**	

Table 1: Experiment results on the random graphs.

Instance	k	NukCP	VNS	ITDBC	mS/B
		max (avg)	max (avg)	max (avg)	max (avg)
uk	2	**5***	**5**	**5**	4
cs4	3	**12**	**12**	**12**	10
email	3	212	**215**	211	210
football	3	**58**	56	**58**	55
polblogs	3	**776**	768	775	**776**
email	4	**651**	648	**651**	
hep-th	4	**344**	338	**344**	

Table 2: Experiment results on the DIMACS benchmark.

We use the same generator method as (Moradi and Balasundaram 2018) to randomly generate 90 instances for k = 2,3,4. These random graphs are divided into 9 families, each of which has 10 instances. As for the DIMACS benchmark[5], we collect all DIMACS instances used in the previous literature for the MkCP. Thus, we select a total of 22 DIMACS instances. We consider massive real-world graphs from the Network Data Repository (Rossi and Ahmed 2015) and among them we choose 65 graphs with more than 10^5 vertices and more than 10^6 edges in this work, which has been widely used into testing different graph problems (Cai et al. 2020; Wang et al. 2020a,b)

For each instance, VNS and NukCP are executed 10 times with random seeds from 1 to 10 while mS/B and ITDBC only execute one time since the random seeds do not affect them. The time limit of all algorithms is set to 1000 seconds for the random and DIMACS benchmarks, while the time limit is 3600 seconds for the massive graphs. 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. For random graphs, we report for each family the average value of max, denoted as $\overline{max}$. If an algorithm fails to provide a solution for an instance, then the corresponding column is marked as "N/A". If an algorithm proves the optimal solution, the corresponding column is marked with a "*". The bold values indicate the best solution value among the different algorithms. Due to space

[2]mS_IP and mB_IP (2014b) are specialized for 2-club and 3-club, respectively. We use mS/B to denote these two algorithms.

[3]https://www.ibm.com/products/software

[4]http://www.gurobi.com

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

Instance	k=2				k=3			
	NukCP	VNS	ITDBC	mS_IP	NukCP	VNS	ITDBC	mB_IP
	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)
bn-human-BNU_1_0*5_s*n_1-bg	**8255**	8081	8081	8227	**15101**	N/A	8081	**15101**
bn-human-BNU_1_0*5_s*n_2-bg	**7494**	7432	7432	7441	**12117**(12034)	N/A	7432	12056
ca-coauthors-dblp	**3300***	**3300**	**3300**	**3300**	**7098**	4369	3300	6652
ca-dblp-2012	**344***	**344**	**344**	**344**	**2136**	608	**2136**	**2136**
ca-hollywood-2009	**11468**	**11468**	N/A	**11468**	**17777**	N/A	N/A	**17777**
channel-500x100x100-b050	**19**	**19**	**19**	**19**	**44**	30	**44**	**44**
dbpedia-link	**293749**	N/A	N/A	**293749**	**369531**	N/A	N/A	**369531**
delaunay_n22	**24**	**24**	**24**	**24**	**52**	38	**52**	**52**
delaunay_n23	**29**	**29**	**29**	**29**	**62**	48	**62**	**62**
delaunay_n24	**27**	**27**	**27**	**27**	**58**	46	**58**	**58**
friendster	**3005**	N/A	N/A	**3005**	**5932**	N/A	N/A	**5932**
hugebubbles-00020	**5**	4	**5**	4	**7**	6	**7**	**7**
hugetrace-00010	**5**	4	**5**	4	**7**	6	**7**	**7**
hugetrace-00020	**5**	4	**5**	4	**7**	6	**7**	**7**
inf-europe_osm	**14**	**14**	**14**	**14**	**23(20.5)**	15	14	**18**
inf-germany_osm	**14**	**14**	**14**	**14**	**23**	15	14	**18**
inf-roadNet-CA	**13***	**13**	**13**	**13**	**17**	16	**17**	**17**
inf-roadNet-PA	**10***	**10**	**10**	**10**	**15***	14	**15**	**15**
inf-road-usa	**10***	**10**	**10**	**10**	**16**	12	10	**16**
rec-dating	**33412**	N/A	N/A	**33412**	**54051**	N/A	N/A	**54051**
rec-epinions	**158933**	N/A	N/A	**158933**	**191728**	N/A	N/A	**191728**
rec-libimseti-dir	**33390**	N/A	N/A	**33390**	**56801**	N/A	N/A	**56801**
rgg_n_2_23_s0	**41**	**41**	**41**	**41**	**65**	55	**65**	**65**
rgg_n_2_24_s0	**42**	41	41	**42**	**75**	58	**75**	**75**
rt-retweet-crawl	**5071***	**5071**	**5071**	**5071**	**6499**	5732	5071	6338
sc-ldoor	**77**	**77**	**77**	**77**	**133**	112	**133**	130
sc-msdoor	**77**	**77**	**77**	**77**	**126**	**126**	**126**	**126**
sc-pwtk	**180***	**180**	**180**	**180**	**214**	**214**	**214**	**214**
sc-rel9	**168**	**168**	**168**	**168**	**266**	N/A	N/A	**266**
sc-shipsec1	**71**	**71**	**71**	**71**	**162**	160	**162**	**162**
sc-shipsec5	**90**	**90**	**90**	**90**	**187**	**187**	**187**	**187**
soc-buzznet	**64290***	**64290**	**64290**	**64290**	**72280**(72279)	N/A	N/A	72276
soc-delicious	**3217***	**3217**	**3217**	**3217**	**6369**(6317.1)	4244	3217	5465
soc-digg	**17644***	**17644**	**17644**	**17644**	**27771**(27680)	N/A	N/A	27653
soc-dogster	**46504**	N/A	N/A	**46504**	**70507**	N/A	N/A	**70507**
socfb-A-anon	**4916***	**4916**	**4916**	**4916**	**8903**	N/A	N/A	**8903**
socfb-B-anon	**4357**	**4357**	**4357**	**4357**	**7402**(7153)	N/A	N/A	6946
socfb-uci-uni	**4961***	N/A	N/A	**4961**	**11088**(7378.2)	N/A	N/A	6928
soc-flickr	**4370**	**4370**	**4370**	**4370**	**9976**(9883)	N/A	N/A	9923

Table 3: Experiment results on the massive graphs I.

limitations, the detailed results as well as the source code of our NukCP can be found in the supplementary material[6].

Tables 1 and 2 summarize the results of the random and DIMACS benchmarks, respectively. Most instances are so easy that all algorithms find the same quality values. We do not report the detailed results of these instances in Tables 1 and 2. For three instance families in the random graphs, NukCP finds better solutions. As for the DIMACS benchmark, only for $email$ with k = 3, NukCP fails to find the same solution as VNS. Moreover, for 75 out of 270 random graphs and 30 out of 66 DIMACS instances, NukCP can prove the optimal solution.

[6]https://github.com/yiyuanwang1988/NukCP.git

The results on the massive graphs are presented in Tables 3 and 4, where we only present the results for k = 2,3, and the results for k = 4 can be found in the supplementary material. NukCP performs better on almost all instances except two instances where mS/B finds better solutions. Among these instances, our NukCP can prove the optimal solution for 44 out of 195 instances, and most of them are concentrated in the case of k = 2. Also, the results show that the performance of our NukCP algorithm becomes a bit worse as the value of k increases. This is because when k has a large value, the size of solution is also large and thus NukCP costs a lot of time to calculate the upper bound as well as maintain the solution.

We compare the average run time of these four algorithms

Instance	k=2				k=3			
	NukCP	VNS	ITDBC	mS_IP	NukCP	VNS	ITDBC	mB_IP
	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)	max (avg)
soc-flickr-und	**27237**	N/A	N/A	**27237**	**40654**	N/A	N/A	**40654**
soc-flixster	**1475**	**1475**	**1475**	**1475**	**3891**(3800.1)	N/A	N/A	3709
soc-FourSquare	**106229**	N/A	N/A	**106229**	**106513**	N/A	N/A	**106513**
soc-lastfm	**5151***	**5151**	**5151**	**5151**	**8227**(8196.5)	N/A	N/A	8105
soc-livejournal	**2652**	**2652**	**2652**	**2652**	3517	N/A	N/A	**3683**
soc-livejournal-user-groups	**1053721**	N/A	N/A	**1053721**	**1300141**	N/A	N/A	**1300141**
soc-LiveMocha	**2981**	**2981**	**2981**	**2981**	8182(8133)	N/A	N/A	**8242**
soc-ljournal-2008	**19433**	N/A	N/A	**19433**	**25624**	N/A	N/A	**25624**
soc-orkut	**27467**	N/A	N/A	**27467**	**50600**	N/A	N/A	**50600**
soc-orkut-dir	**33314**	N/A	N/A	**33314**	**59375**	N/A	N/A	**59375**
soc-pokec	**14855***	**14855**	**14855**	**14855**	**16415**(16414.7)	N/A	N/A	16289
soc-sinaweibo	**278490**	N/A	N/A	**278490**	**382513**	N/A	N/A	**382513**
soc-twitter-higgs	**51387**	N/A	N/A	**51387**	**78697**	N/A	N/A	**78697**
soc-youtube	**25410***	**25410**	**25410**	**25410**	**33636**(33538)	N/A	N/A	33413
soc-youtube-snap	**28755***	**28755**	**28755**	**28755**	**41215**(40864)	N/A	N/A	40605
tech-as-skitter	**35456**	**35456**	N/A	**35456**	**57395**(57369)	N/A	N/A	57349
tech-ip	**1833162**	N/A	N/A	**1833162**	**1855649**	N/A	N/A	**1855649**
twitter_mpi	**532053**	N/A	N/A	**532053**	**765315**	N/A	N/A	**765315**
web-arabic-2005	**1103***	**1103**	**1103**	**1103**	**1137***	**1137**	**1137**	**1137**
web-baidu-baike	**97849**	N/A	N/A	**97849**	**166176**	N/A	N/A	**166176**
web-it-2004	**470***	**470**	**470**	**470**	**1086***	482	**1086**	**1086**
web-uk-2005	**851***	**851**	**851**	**851**	**1350***	**1350**	**1350**	**1350**
web-wikipedia_link	**825148**	N/A	N/A	**825148**	**1064494**	N/A	N/A	**1064494**
web-wikipedia2009	**2625***	**2625**	**2625**	**2625**	**3183**(2858.1)	2630	2625	2685
web-wikipedia-growth	**226074***	N/A	N/A	**226074**	**302564**(2747.9)	N/A	N/A	**302564**
wikipedia_link_en	**68873**	N/A	N/A	**68873**	**80686**	N/A	N/A	**80686**

Table 4: Experiment results on the massive graphs II.

benchmark	k	NukCP	VNS	ITDBC	mS/B
random graphs	2	0.01	0.01	0.01	0.01
	3	0.01	0.06	0.02	0.01
	4	0.08	2.4	1.3	
DIMACS	2	0.01	49.98	6.28	0.01
	3	9.09	46.77	40.52	29.43
	4	6.77	18.72	9.59	
massive graphs	2	33.82	1288.89	1230.22	71.33
	3	912.61	3054.07	2375.97	1229.91
	4	2071.4	3465.55	2436.12	

Table 5: Average run time for all benchmarks

on all benchmarks (Table 5), where the run time of each run of an algorithm is the time to reach the final solution. Figure 3 displays the average run time of NukCP and the corresponding competitor when both algorithms find the same maximal and average solution values, which further indicates the superiority of NukCP, with a few exceptions.

To further verify the effectiveness of the proposed NukCP, we also evaluate the performance of NukCP on two popular benchamarks, including Stanford Large Network Dataset Collection[7] and DIMACS10 [8]. Due to space limitations, we present the detailed results of NukCP and all competitors in the supplementary material.

Benchmark	TRIM		DRM	
	$r\%$	time(s)	$r\%$	time(s)
random graphs	44.96%	<0.01	43.97%	<0.01
DIMACS	33.75%	0.03	33.68%	0.01
massive graphs	57.72%	844.92	58.77%	376.43

Table 6: Reduction efficiency of TRIM and DRM.

The Effectiveness of the Proposed Strategies

Table 4 reports the reduction ratio ($r\%$) and the time consumption ($time$) of TRIM (2018) and DRM for the initial reduction on all benchmarks with the same bound functions. The difference of the reduction ration between DRM and TRIM is obvious on the massive graphs. This is because TRIM costs too much time on calculating upper bound values and fails to delete any vertices.

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

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

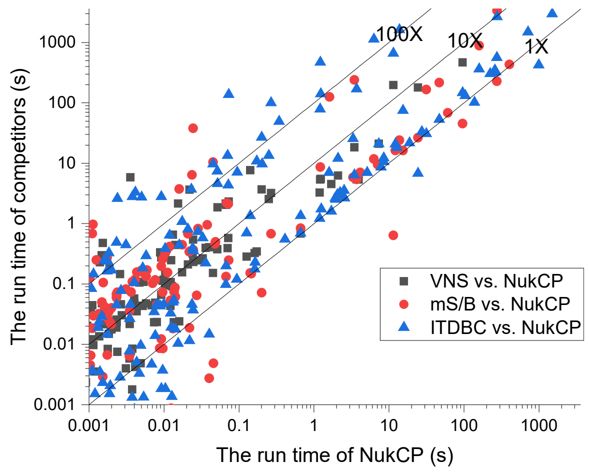

Figure 3: Average run time of NukCP and competitors.

Benchmark	vs. NukCP1		vs. NukCP2	
	#better	#worse	#better	#worse
random graphs	1	1	1	1
DIMACS	1	0	1	0
massive graphs	15	4	18	3

Table 7: Compare NukCP with two modified versions on all benchmarks. #better and #worse denote the number of instances where NukCP finds better and worse results, respectively.

To verify the effectiveness of STCC, we design two alternative algorithms where NukCP1 utilizes DCC (2020) instead of STCC and NukCP2 utilizes BoundedCC (2021) instead of STCC. The results in Table 5 show that our proposed STCC plays a key role in the NukCP algorithm and performs well in the massive graphs.

Conclusion

In this paper, we propose an efficient reduction strategy and a variant of configuration checking strategy for the MkCP. Based on the above strategies, we develop a local search algorithm called NukCP. Experiments show NukCP significantly outperforms the state-of-the-art heuristic algorithms. As we know, it is the first work for solving the MkCP on the massive graphs, and thus the proposed NukCP can help establish such a standard for the MkCP. As for future work, STCC can be considered as a general idea to solve many other optimization problems with connectivity constraints.

Acknowledgements

This work was supported by National Key R&D Program of China (2020AAA0105200), NSFC under Grant No. (61806050, 61972063, 61976050), the Fundamental Research Funds for the Central Universities (2412020FZ030, 2412018QD022), and Jilin Science and Technology Association (QT202005).

References

Almeida, M. T.; and Carvalho, F. D. 2012. Integer models and upper bounds for the 3-club problem. *Networks*, 60(3): 155–166.

Almeida, M. T.; and Carvalho, F. D. 2014a. An analytical comparison of the LP relaxations of integer models for the k-club problem. *European Journal of Operational Research*, 232(3): 489–498.

Almeida, M. T.; and Carvalho, F. D. 2014b. Two-phase heuristics for the k-club problem. *Computers & operations research*, 52: 94–104.

Balasundaram, B.; Butenko, S.; and Trukhanov, S. 2005. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1): 23–39.

Bourjolly, J.-M.; Laporte, G.; and Pesant, G. 2000. Heuristics for finding k-clubs in an undirected graph. *Computers & Operations Research*, 27(6): 559–569.

Bourjolly, J.-M.; Laporte, G.; and Pesant, G. 2002. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1): 21–28.

Butenko, S.; and Wilhelm, W. E. 2006. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1): 1–17.

Cai, S.; Hou, W.; Wang, Y.; Luo, C.; and Lin, Q. 2020. Two-goal Local Search and Inference Rules for Minimum Dominating Set. In *IJCAI*, 1467–1473.

Cai, S.; and Lin, J. 2016. Fast solving maximum weight clique problem in massive graphs. In *IJCAI*, 568–574.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10): 1672–1696.

Cavique, L.; Mendes, A. B.; and Santos, J. M. 2009. An algorithm to discover the k-clique cover in networks. In *Portuguese Conference on Artificial Intelligence*, 363–373.

Chang, M.-S.; Hung, L.-J.; Lin, C.-R.; and Su, P.-C. 2013. Finding large k-clubs in undirected graphs. *Computing*, 95(9): 739–758.

Chen, J.; Cai, S.; Pan, S.; Wang, Y.; Lin, Q.; Zhao, M.; and Yin, M. 2021. NuQClq: an effective local search algorithm for maximum quasi-clique problem. In *AAAI*, 12258–12266.

Chen, P.; Wan, H.; Cai, S.; Li, J.; and Chen, H. 2020. Local search with dynamic-threshold configuration checking and incremental neighborhood updating for maximum k-plex problem. In *AAAI*, 2343–2350.

Gao, J.; Chen, J.; Yin, M.; Chen, R.; and Wang, Y. 2018. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *IJCAI*, 1449–1455.

Goodreau, S. M.; Kitts, J. A.; and Morris, M. 2009. Birds of a feather, or friend of a friend? Using exponential random graph models to investigate adolescent social networks. *Demography*, 46(1): 103–125.

Jia, S.; Gao, L.; Gao, Y.; Nastos, J.; Wen, X.; Huang, X.; and Wang, H. 2018. Viewing the meso-scale structures in protein-protein interaction networks using 2-clubs. *IEEE Access*, 6: 36780–36797.

Jiang, H.; Li, C.-M.; and Manyà, F. 2016. Combining Efficient Preprocessing and Incremental MaxSAT Reasoning for MaxClique in Large Graphs. In *ECAI*, 939–947.

Moradi, E.; and Balasundaram, B. 2018. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, 12(8): 1947–1957.

Ouyang, Q.; Kaplan, P. D.; Liu, S.; and Libchaber, A. 1997. DNA solution of the maximal clique problem. *Science*, 278(5337): 446–449.

Pajouh, F. M.; and Balasundaram, B. 2012. On inclusion-wise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization*, 9(2): 84–97.

Pattillo, J.; Youssef, N.; and Butenko, S. 2013. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1): 9–18.

Rossi, R. A.; and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *AAAI*, 4292–4293.

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

Shahinpour, S.; and Butenko, S. 2013b. Distance-based clique relaxations in networks: s-clique and s-club. *Models, algorithms, and technologies for network analysis*, 149–174.

Veremyev, A.; Boginski, V.; Pasiliao, E. L.; and Prokopyev, O. A. 2021. On integer programming models for the maximum 2-club problem and its robust generalizations in sparse graphs. *European Journal of Operational Research*, 1–16.

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2018. A Fast Local Search Algorithm for Minimum Weight Dominating Set Problem on Massive Graphs. In *IJCAI*, 1514–1522.

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2020a. SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280: 103230.

Wang, Y.; Cai, S.; Pan, S.; Li, X.; and Yin, M. 2020b. Reduction and local search for weighted graph coloring problem. In *AAAI*, volume 34, 2433–2441.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for Maximum Weight Clique problem. In *AAAI*, 805–811.

Zhou, Y.; Hu, S.; Xiao, M.; and Fu, Z.-H. 2021. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *AAAI*, volume 35, 12453–12460.