Local Search with Dynamic-Threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-plex Problem

Peilin Chen,[1] **Hai Wan,**[1*] **Shaowei Cai,**[2*] **Jia Li,**[1] **Haicheng Chen**[1]
[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
wanhai@mail.sysu.edu.cn, {chenpl7, lijia49, chenhch8}@mail2.sysu.edu.cn, caisw@ios.ac.cn

Abstract

The Maximum k-plex Problem is an important combinatorial optimization problem with increasingly wide applications. In this paper, we propose a novel strategy, named Dynamic-threshold Configuration Checking (DCC), to reduce the cycling problem of local search. Due to the complicated neighborhood relations, all the previous local search algorithms for this problem spend a large amount of time in identifying feasible neighbors in each step. To further improve the performance on dense and challenging instances, we propose Double-attributes Incremental Neighborhood Updating (DINU) scheme which reduces the worst-case time complexity per iteration from $O(|V| \cdot \Delta_G)$ to $O(k \cdot \Delta_{\overline{G}})$. Based on DCC strategy and DINU scheme, we develop a local search algorithm named DCCplex. According to the experiment result, DCCplex shows promising result on DIMACS and BHOSLIB benchmark as well as real-world massive graphs. Especially, DCCplex updates the lower bound of the maximum k-plex for most dense and challenging instances.

Introduction

In social network analysis, detecting a large cohesive subgraph is a fundamental and extensively studied topic with various applications. Clique is a classical and ideal model in the field of cohesive subgraph detection. The Maximum Clique Problem (MCP), that is, to find a complete graph of maximum size in a given graph, is a fundamental problem in graph theory and finds wide application in many fields, such as biochemistry and genomics (Butenko and Wilhelm 2006; Pullan 2007), wireless network (Lakhlef 2015; Luo et al. 2015), data mining (Boginski, Butenko, and Pardalos 2006; Conte et al. 2018) and many others.

However, in some real-world applications, the networks of interest may be built based on empirical data with noises and faults. In these cases, large cohesive subgraphs hardly appear as ideal cliques. To tackle this problem, many clique relaxation models have been proposed (Luce 1950; Mokken 1979; Pattillo et al. 2013). In this paper, we focus on k-plex, a degree-based clique relaxation model. A simple undirect graph with n vertices is a k-plex if each vertex of this graph has at least $n - k$ neighbors. The *maximum k-plex problem*, that is, to find a k-plex of maximum size on a given graph with a given integer k, has received increasing attention from researchers in the fields of social network analysis and data mining (Kondo and Okubo 2012; Xiao et al. 2017; Conte et al. 2018; Gao et al. 2018).

As with MCP, the maximum k-plex problem is known to be NP-hard (Balasundaram, Butenko, and Hicks 2011). Practical methods for this problem can be mainly categorized into exact algorithms and heuristic ones. Balasundaram, Butenko, and Hicks (2011) proposed a branch-and-bound algorithm based on a polyhedral study of this problem. McClosky and Hicks (2012) developed two branch-and-bound algorithms adapted from combinatorial clique algorithms. Recently, Xiao et al. (2017) proposed an exact algorithm which breaks the trivial exponential bound of 2^n for maximum k-plex problem with $k \geq 3$. Gao et al. (2018) proposed several graph reduction methods and integrated them into a brand-and-bound algorithm.

However, these exact methods do not scale well and several heuristic approaches, mainly local search (LS) ones, have been proposed for solving large and hard instances. Gujjula, Seshadrinathan, and Meisami (2014) proposed a hybrid metaheuristic based on the GRASP method. Miao and Balasundaram (2017) improved the construction procedure to provide a better initial solution for GRASP method. Zhou and Hao (2017) developed a tabu search algorithm named FD-TS which achieved state-of-the-art performance.

Though LS algorithms have made some achievement on this challenging problem, it has some limitations, a major one of which is the cycling problem (Hoos and Stützle 2004). When the search process is cycling, it is stuck in local optima and frequently visits a small group of a candidate solution. For a long time, much effort has been devoted to reducing the cycling problem in local search. Tabu search (Glover and Laguna 1998) maintains a short-term memory of the recent search steps and forbid reversing the recent changes. Configuration Checking (Cai, Su, and Sattar 2011) remembers local state change and reduces global cycling problem by prohibiting cycling locally.

Different from the tabu method, Configuration Checking

[*]corresponding author and having equal contribution with the first author

(CC) is a non-parameter strategy which exploits the circumstance information to reduce cycling problem in local search. Recently, CC and its variants have been successfully applied in various combinatorial problems (Cai and Su 2013; Wang, Cai, and Yin 2016; Wang et al. 2018). However, as is shown in (Cai and Su 2013), the CC strategy becomes ineffective when solving the dense SAT instances where most variables are connected to each other. The reason is that the forbidding strength of the CC is too weak on instances which have dense variable-variable connections. The situation is the same when applying CC strategy for the maximum k-plex problem on dense graphs. In this paper, we propose Dynamic-threshold Configuration Checking (DCC), to enable an adaptive forbidding strength for CC so that it can handle dense instances well.

In the literature, there exists a lightweight neighborhood evaluation method for the maximum clique problem (Battiti and Protasi 2001). For the maximum k-plex problem , however, the relaxed constraints on connectivity lead to a complicated neighborhood relation, which is the main difficulty when designing an efficient neighborhood updating method. In this paper, we propose Double-attributes Incremental Neighborhood Updating (DINU) scheme to tackle the complicated neighborhood relations for the maximum k-plex problem. Theoretical and empirical analysis show that DINU cuts down the overhead per search step and significantly boosts the iterating speed of local search for the maximum k-plex problem .

Based on the DCC strategy and DINU scheme, we develop an LS algorithm named DCCplex. According to the experiments, DCCplex shows a promising result on the DIMACS benchmark and dominates on the BHOSLIB benchmark, updating the lower bounds of the size of the maximum k-plexes for most hard instances. Besides, DCCplex achieves state of the art performance on massive graphs.

Preliminaries

Basic Definitions and Notations

An undirected graph is defined as $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each edge e consists of two vertices, denoted as $e = (v, u)$, where v and u are the $endpoints$ of this edge. Two vertices are *neighbors* if they belong to an edge and *non-neighbors* otherwise. Let $N(v)$ denote the set of all neighbors of v. The *degree* of vertex v is defined as the $deg(v) = |N(v)|$. The maximum degree of a graph G , denoted by Δ_G, and the minimum degree of a graph, denoted by δ_G, are the maximum and minimum degree of its vertices. The edge density of $G = (V, E)$ is defined as $\rho = \frac{2|E|}{|V|(|V|-1)}$. For a vertex set S, let $N(S) = \bigcup_{v \in S} N(v) \setminus S$ be the set of neighbors of S and $G[S] = (S, E \cap (S \times S))$ be the induced graph of S. The complement graph of $G = (V, E)$ is the graph $\overline{G}$ on the same set of vertices, and with edges set $\{(u, v)|u \neq v, (u, v) \notin E\}$.

Given a graph G and an integer k, a subset $S \subseteq V$ is a k-plex, if $|N(v) \cap S| \geq |S| - k$ for all $v \in S$. A vertex $v \in S$ is a *saturated vertex* if $|N(v) \cap S| = |S| - k$. The *saturated set* $\mathcal{C}[S]$ of set S is the set of all saturated vertices in S.

Local Search for Maximum k-plex Problem

Given a graph $G = (V, E)$ and an integer k, a typical three-phase LS algorithm for the maximum k-plex problem maintains a feasible k-plex $S \subseteq V$ as *candidate solution* and uses three operators, Add, $Swap$ and $Perturb$ to modify it iteratively. Three sets $AddSet(S)$, $SwpSet(S)$ and $PerturbSet(S)$ contain the *objects* of these three operators. They are defined as follows:

$$\begin{aligned} AddSet(S) =& \{v \in N(S) \mid |N(v) \cap S| > |S| - k, \\ & \mathcal{C}[S] \setminus N(v) = \emptyset\} \\ SwapSet(S) =& \{v \in N(S) \mid |N(v) \cap S| \geq |S| - k, \\ & |\mathcal{C}[S] \setminus N(v)| = 1\} \cup \{v \in N(S) : \\ & |N(v) \cap S| = |S| - k, |\mathcal{C}[S] \setminus N(v)| = \emptyset\} \\ PerturbSet(S) =& S \end{aligned}$$

$AddSet(S)$ contains the vertices in $N(S)$ that can be added into S directly. A vertex $v \in N(S)$ is in $AddSet(S)$ if it has less than k non-neighbors in S and is adjacent to all saturated vertices. A vertex $v \in N(S)$ is in $SwapSet(S)$ if it satisfies one of the following conditions, (i) v has k non-neighbors in S and is adjacent to all saturated vertices, (ii) v has less than k non-neighbors in S but is not adjacent to one saturated vertex. A higher priority are given to Add and $Swap$ operator. When $AddSet(S)$ and $SwapSet(S)$ are empty, a vertex in $PerturbSet(S)$ will be removed from S.

Dynamic-threshold Configuration Checking

Configuration Checking (CC) (Cai, Su, and Sattar 2011), is a parameter-free strategy that can exploit the structural property of the problem to reduce cycling problem in local search. The *configuration* of a vertex is defined as the states of its neighbors. The main idea of CC strategy is that if the configuration of a vertex remains unchanged since its last removal from candidate solution, then it is forbidden to be added back into the candidate solution.

Among the variants of CC, we highlight the Strong Configuration Checking (SCC) strategy which was proposed in (Wang, Cai, and Yin 2016) for the Maximum Weight Clique Problem. The difference between SCC and CC is that SCC allows a vertex v to be added into the candidate solution only when some of v's neighbors have been added since v's last removal, while CC allows the adding of a vertex v when some of v's neighbors have been either added or removed.

According to our preliminary experiments, applying CC or SCC directly does not lead to a good performance for the maximum k-plex problem. The reason is that the configurations of the high-degree vertices in these graphs are very likely to change and the forbidding strength of CC and other existing CC variants is too weak on these vertices. Therefore, we propose a new variant of CC named Dynamic-threshold Configuration Checking (DCC), which can dynamically adjust its forbidding strength on vertices with various degrees.

The main idea of the DCC strategy is to increase the forbidding strength on the vertices that are frequently operated. For each vertex, DCC maintains an integer $nChange$ to count the number of changes of its configuration and an integer $threshold$ to control the forbidding strength. A vertex v

is allowed to be added into the candidate solution only when DCC condition $nChange(v) \geq threshold(v)$ is satisfied. The following four rules specify the DCC strategy.

DCC-InitialRule. In the beginning of search process, for all $v \in V$, $nChange(v) = threshold(v) = 1$.

DCC-AddRule. When v is added into the candidate solution, $nChange(v) = 0$, $threshold(v) + +$, and for all $v' \in N(v)$, $nChange(v') + +$.

DCC-SwapRule. When v is added into the candidate solution at the cost of removal of u, $nChange(u) = 0$.

DCC-PerturbRule. When v is removed from candidate solution, $nChange(v) = 0$.

Note that the SCC strategy is a special case of DCC strategy whose $nChange(v)$ is a Boolean value and $threshold(v)$ is fixed to 1. The SCC condition is $nChange(v) = 1$. Lemma 1 illustrate their relation.

Lemma 1. *If a vertex v satisfies the DCC condition, then it satisfies the SCC condition. The reverse is not necessarily true.*

Proof. According to DCC rules, $threshold(v) \geq 1$ for $\forall v \in V$ during the search processs. If the DCC condition $nChange(v) \geq threshold(v)$ holds, then $nChange(v) \geq threshold(v) \geq 1$. So at least one neighbor of v must be added into candidate solution since the last time v was removed. So the SCC condition is satisfied.

Suppose v satisfies the SCC condition $nChange(v) = 1$, but $threshold(v) > nChange(v)$. In this case the DCC condition is not satisfied. □

According to Lemma 1, we can conclude that DCC has stronger forbidding strength than SCC. Note that some works exist that tried to quantify the CC strategy (Luo, Su, and Cai 2012; Wang et al. 2019). But we are the first to quantify the CC strategy to enable a dynamic forbidding strength.

An Incremental Neighborhood Updating Scheme

The performance of an LS algorithm is influenced not only by its search strategies but also the implementation. Incremental neighborhood updating scheme is an important speed-up technique for LS, which is verified in (Battiti and Protasi 2001; Cai, Luo, and Su 2015; Lin et al. 2019). Compared with evaluating neighbors from scratch, evaluating them incrementally can significantly reduce the computational overhead per search step.

Note that each vertex in $AddSet(S)$ or $SwapSet(S)$ corresponds to a feasible neighbor of S. Which set a vertex $v \in N(S)$ belongs to depends on not only the number of non-neighbors in S but also non-neighbors in $\mathcal{C}[S]$. For the maximum 1-plex (clique) problem, all the vertices in S are saturated vertices. However, for $k \geq 2$, S and $\mathcal{C}[S]$ are not equal in general. How to keep track of the saturated set and its connection with vertices in $N(S)$ is the major difficulty to design an efficient neighborhood updating scheme. The straightforward neighborhood updating method used in FD-TS and other LS methods for this problem contains two parts: 1) scan all the vertices in S, identify all the saturated vertices, 2) for each $v \in N(S)$, scan all the neighbors of v and add them into the corresponding sets according to the numbers of non-neighbors in S and $\mathcal{C}[S]$. Obviously its worst-case time complexity is $O(|S| \cdot \Delta_G) + O((|V| - |S|) \cdot \Delta_G) = O(|V| \cdot \Delta_G)$. It is time-consuming on dense graphs.

In this section, we propose a novel neighborhood updating scheme named DINU (Double-attributes Incremental Neighborhood Updating) to tackle the complicated neighborhood relations for maximum k-plex problem .

Implementation Details of DINU

Data Structures Before presenting the details of DINU, we first introduce two attributes for each vertex.

(1)$NNS = |S \setminus N(v)|$, i.e., the number of non-neighbors in the candidate solution S.

(2)$NNC = |\mathcal{C}[S] \setminus N(v)|$, i.e., the number of non-neighbors in the saturated set $\mathcal{C}[S]$.

With these two attributes, we redifine the $AddSet(S)$ and $SwapSet(S)$ as follows:

$$\begin{aligned} AddSet(S) &= \{v \in N(S) \mid NNS(v) \leq k-1, NNC(v) = 0\} \\ SwapSet(S) &= \{v \in N(S) \mid NNS(v) \leq k, NNC(v) = 1\} \\ &\cup \{v \in N(S) \mid NNS(v) = k, NNC(v) = 0\}. \end{aligned}$$

Figure 1 shows in an intuitive way how to classify a vertex according to its NNS and NNC. Note that for MCP, all vertices in the candidate solution are saturated vertices, thus $NNS(v) = NNC(v)$ for all $v \in V$.

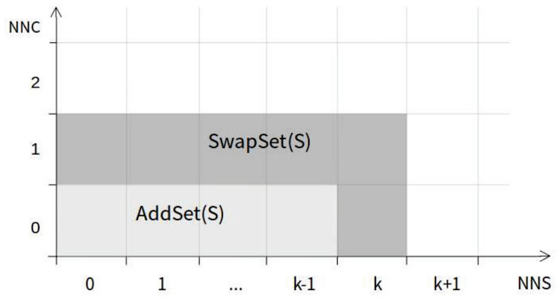

Figure 1: The Regions of $AddSet(S)$ and $SwapSet(S)$ in $NNS - NNC$ Coordinates

Updating Rules The underlying idea of DINU is to keep track of the NNS and NNC attributes of each vertex and update the set a vertex belongs to only when it crosses the border in $NNS - NNC$ coordinates. Consider a situation where a vertex v is added into candidate solution. The following rules specify how to update the set a vertex belongs to according to the change of its attributes.

- In each search step, after a vertex v is added into S, for all $u \in N_{\overline{G}}(v)$, $NNS(u) + +$. If $NNS(u) = k - 1$, then for all $w \in N_{\overline{G}}(u)$, $NNC(w) + +$.
- If $NNC(v)$ = 0 and $NNS(v)$ changes from $k - 1$ to k, move x from $AddSet(S)$ to $SwapSet(S)$.
- If $NNC(v) \leq 1$ and $NNS(v)$ changes from k to $k + 1$, remove x from $SwapSet(S)$.

- If $NNS(v) \leq k-1$, $NNC(v)$ changes from 0 to 1, move x from $AddSet(S)$ to $SwapSet(S)$.
- If $NNS(v) \leq k$, $NNC(v)$ changes from 1 to 2, remove x from $SwapSet(S)$.

With the above rules, we can incrementally update the $AddSet(S)$ and $SwapSet(S)$ after Add operations. The updating rules for Perturb operations can be got similarly and are detailed in the supplemental material.

Implementation in Add Operation We implement the updating rules in the DINU-ADD function, whose details is presented in Algorithm 1.

Algorithm 1: DINU-ADD(S, v)

```
Input  : Current $S$, a vertex $v$ to be added
Output: New $AddSet(S)$ and $SwapSet(S)$
1  forall $u \in N_{\overline{G}}(v) \setminus S$ do
2      $NNS(u)++$;
3      move $u$ according to the updating rules;
4  forall $u \in N_{\overline{G}}(v) \cap S$ do
5      $NNS(u)++$;
6      if $NNS(u) = k-1$ then
7          forall $w \in N_{\overline{G}}(u)$ do
8              $NNC(w)++$;
9              if $w \notin S$ then
10                 move $w$ according to the updating rules;
11 if $NNS(v) = k-1$ then
12     forall $m \in N_{\overline{G}}(v)$ do
13         $NNC(m)++$;
14         if $m \notin S$ then
15             move $m$ according to the updating rules;
16 return $AddSet(S)$ and $SwapSet(S)$
```

After adding v into S, we increase the NNS value of its non-neighbors out of S by one (line 2) and move them according to updating rules (line 3). The non-neighbors in S whose NNS is increased to $k-1$ become saturated vertices. The non-neighbors of these saturated vertices have to increase their NNC value by one and make a move according to Update-Rules (line 5-10). In the end, if the added vertex v is a saturated vertex, increase the NNC value of all its non-neighbors as described above (line 11-15) and move them if necessary. The new DINU-Perturb function is easy to get by analogy and is omitted for saving space.

Complexity Analysis

Next, we will analyze the complexity of DINU in *Add* operation. We give the following theorem.

Theorem 1. *With the DINU scheme, each search step of local search algorithm for maximum k-plex problem has a worst-case complexity of* $O(k \cdot \Delta_{\overline{G}})$.

Proof. We begin with the analysis of new DINU-ADD function. When a vertex v is added to S, first, for all $u \in N_{\overline{G}}(v) \setminus S$, $NNS(u)++$, operations of this loop (line 1-3) are at most $\Delta_{\overline{G}}$. Second, for all $u \in N_{\overline{G}}(v) \cap S$, $NNC(u)++$. If u is a saturated vertex, then we need to update NNC for all $w \in N_{\overline{G}}(u)$, $|N_{\overline{G}}(u)|$ is at most $\Delta_{\overline{G}}$ and move operation costs $O(1)$. According to the definition of k-plex, we have $|N_{\overline{G}}(v) \cap S| \leq k-1$, so the time cost of the loop in line 4-10 is bound by $(k-1) \cdot \Delta_{\overline{G}}$. At last, if this added vertex v is a saturated vertex, the number of operations inside this loop (line 11-15) are at most $\Delta_{\overline{G}}$. Similarly, we can get the same time complexity of the new DINU-PERTURB function. In practice, DINU-SWAP function can be implemented by combining a DINU-PERTURB and DINU-ADD function. Therefore, we can conclude that the worst-case complexity of each iteration is $O(k \cdot \Delta_{\overline{G}})$. □

For a small constant k and a dense graph G where $\Delta_{\overline{G}}(v)$ is much smaller than $\Delta_G(v)$, our scheme has a lower worst-case complexity compared with the straightforward method, which will be verified by experiments.

DCCplex Algorithm

Based on the DCC strategy and the DINU scheme, we develop an LS algorithm named DCCplex, whose pseudocode is shown in Algorithm 2. Initially, the best found k-plex, denoted as S^*, is initialized as an empty set. In each loop (line 2-10), an initial solution is firstly constructed (line 3) as the starting point of the search trajectory, and the search procedure starts. If the best solution in this search trajectory S_{lbest} is better than the best solution ever found S^*, S^* is updated by S_{lbest} and PEEL function (line 8) is called to reduce the graph. If the reduced graph has fewer vertices than $|S^*|$, then $|S^*|$ is returned as one of the optimum solutions. Three major components in DCCplex are initial solution construction, search procedure and graph peeling. We describe them in details as follows.

Algorithm 2: DCCplex algorithm

```
Input  : A graph $G$, an integer $k$, cutoff time $ct$,
         iterations limit $L$
Output: The largest $k$-plex found
1  $S^* \leftarrow \emptyset$;
2  while $elapsedtime < ct$ do
3      $S \leftarrow$ INITCONSTRUCT$(G, k)$;
4      $\rho \leftarrow$ compute the edge density of $G$;
5      $S_{lbest} \leftarrow$ SEARCH$(G, k, S, L, \rho)$;
6      if $|S_{lbest}| > |S^*|$ then
7          $S^* \leftarrow S_{lbest}$;
8          $G \leftarrow$ PEEL$(G, k, |S^*|)$ ;
9      if $|V| \leq |S^*|$ then
10         return $S^*$;
11 return $S^*$
```

We adopt the construction function in FD-TS (Zhou and Hao 2017). The INITCONSTRUCT function starts with an empty initial solution S and repeatedly add the vertex that is operated least frequently (breaking ties randomly) in

$AddSet(S)$ into S until $AddSet(S)$ is empty. Then the final solution S is returned as the initial solution. By giving priority to vertices that are operated less frequently, the construction procedure can generate diversified initial solutions in different rounds.

Algorithm 3: SEARCH(G, k, S, L, ρ)

```
Input  : A graph $G$, an integer $k$, initial solution $S$,
         search depth $L$, edge density $\rho$
Output: The largest $k$-plex in this search procedure
1  $curStep \leftarrow 0$, $S_{lbest} \leftarrow S$,;
2  $nChange(v) \leftarrow 1, threshold(v) \leftarrow 1$ for all $v \in V$;
3  initialize DINU data structures ;
4  obtain $AddSet(S), SwapSet(S)$ ;
5  while $curStep < L$ do
6      if $AddSet(S) \neq \emptyset$ then
7          add a vertex $v \in AddSet(S)$ into $S$;
8      else if $SwapSet(S) \neq \emptyset$ then
9          swap a vertex $u \in S$ for a vertex
             $v \in SwapSet(S)$;
10     else
11         remove a random vertex from $S$;
12     update $nChange$ and $threshold$ according to the
         DCC rules;
13     if $\rho > \rho_0$ then
14         update $AddSet(S), SwapSet(S)$ with DINU;
15     else
16         obtain $AddSet(S), SwapSet(S)$ from scratch ;
17     if $|S| > |S_{lbest}|$ then
18         $Slbest \leftarrow S$;
19     $curStep \leftarrow curStep + 1$
20 return $S_{lbest}$;
```

The SEARCH function iteratively modifies the candidate solution until the iterations limit L is reached, as is shown in Algorithm 3. The S_{lbest} records the best-quality solution in the search process so far. Before the search starts, the algorithm will initialize DINU data structures and $AddSet(S), SwapSet(S)$. If the $AddSet(S)$ is not empty, then the algorithm selects a vertice in $AddSet(S)$ and add it into S. If the $AddSet(S)$ is empty the $SwapSet(S)$ is not, the algorithm selects a vertice in $SwapSet(S)$ and add it into S after remove a vertex from S. If both sets are empty, then a vertex randomly selected is removed from S to create a new search area. After each search step, if the edge density is more than ρ_0, $AddSet(S)$ and $SwapSet(S)$ are updated with DINU scheme. Otherwise, they are obtained from scratch. If $|S| > |S_{lbest}|$, then record S in S_{lbest}.

If the S_{lbest} returned by SEARCH is better than S^*, then S^* is updated with S_{lbest} and the PEEL function is called to recursively deletes the vertices (and their incident edges) with a degree less than $|S^*| - k + 1$ until no such vertex exists. It is sound to remove these vertices since they can not be included in any feasible k-plexes larger than $|S^*|$.

Experimental Result

Experiment Preliminaries

In this section, we carry out extensive experiments on DIMACS and BHOSLIB (Xu et al. 2005) benchmark as well as massive graphs. We choose two state of the art algorithms as competitors, including a heuristic one FD-TS (Zhou and Hao 2017) and an exact one BnB (Gao et al. 2018). All of these algorithms are all implemented in C++ and compiled by g++ with '-O3' option. All experiments are run on an Intel Xeon CPU E7-4830 v3 @ 2.10GHz with 128 GB RAM server under Ubuntu 16.04.5 LTS. We set the search depth $L = 4000$ and the $\rho_0 = 0.5$ for DCCplex. The All algorithms are executed 25 independently times with different random seeds on each graph with $k = 2, 3, 4$.

Evaluation on DIMACS and BHOSLIB

We evaluate DCCplex and FD-TS on DIMACS and BHOSLIB benchmark with a cutoff time of $1000s$. Note that most DIMACS instances are so easy that FD-TS and DCCplex find the same quality k-plex quickly, and thus are not reported. According to our preliminary experiments, the remaining instances are so challenging for BnB that it returns a much smaller k-plex than its heuristic competitors even given 10000 seconds. Thus we do not report the result of BnB.

According to Table 1, DCCplex dominates on C domain, MANN domain and keller domain in DIMACS benchmark as well as the whole BHOSLIB benchmark. DCCplex updates the best-known solutions for 7 instances and 45 instances in DIMACS and BHOSLIB benchmark respectively. Note that these instances are large and challenging. For some instances in brock domain and san domain, FD-TS is more robust and achieves better average performance, partly because of its more random behavior in adding and swapping phases (Zhou and Hao 2017). The complementarity between DCCplex and FD-TS indicates that it is promising to build an algorithm-selector to achieve generally good performance across the wide range of problem domains in DIMACS benchmark.

In BHOSLIB benchmark, DCCplex is always better than FD-TS in terms of maximum size and average size, except one instance (frb53-24-4 with $k = 2$). It is worthy to note that for the very challenging instance frb100-40, DCCplex reaches a much larger k-plex than the state of the art algorithms, which is a great progress for this problem.

Evaluation on Massive Graphs

We evaluate DCCplex on real-world massive graphs from Network Data Repository online (Rossi and Ahmed 2015), which have recently been used in testing the performance of local search algorithms (Rossi et al. 2014; Wang, Cai, and Yin 2016; Cai, Lin, and Luo 2017). We only test on 36 graphs that have more than 100000 vertices. According to our experiments, within the cutoff time of $1000s$, BnB returns a solution and proves its optimality on 76 out of 108 ($= 36 \times 3$) instances. For these 76 instances, DCCplex can achieve the same quality solution within the cutoff time of $100s$ (most less than $10s$). As for the remaining 32 instances,

Table 1: Evaluation on DIMACS and BHOSLIB with $k = 2, 3, 4$

Instance	k=2		k=3		k=4	
	FD-TS	DCCplex	FD-TS	DCCplex	FD-TS	DCCplex
C1000.9	82(81.2)	82(**81.88**)	95(94.8)	95(**95**)	108(107.48)	**109(108)**
C2000.5	20(19.44)	20(**19.68**)	23(22.04)	23(**22.2**)	25(25)	25(25)
C2000.9	92(90.16)	**93(92.04)**	106(104.84)	**108(107.12)**	120(118.16)	**123(121.24)**
C4000.5	21(20.24)	**22(20.92)**	24(23.28)	24(**23.6**)	27(26.12)	27(**26.28**)
DSJC1000.5	18(18)	18(18)	21(21)	21(21)	24(**24**)	24(23.96)
MANN_a81	2162(2161.92)	2162(**2162**)	3240(3240)	3240(3240)	3240(3240)	3240(3240)
brock400_4	33(32.84)	33(32.84)	36(36)	36(36)	41(41)	41(41)
brock800_1	25(25)	25(25)	30(**29.8**)	30(29.16)	34(**34**)	34(33.96)
brock800_2	25(25)	25(25)	30(30)	30(30)	34(**33.72**)	34(33.28)
brock800_3	25(25)	25(25)	30(30)	30(30)	34(**34**)	34(33.84)
brock800_4	26(**26**)	26(25.6)	29(29)	29(29)	**34(33.12)**	33(33)
gen400_p0.9_65	74(**74**)	74(73.08)	101(101)	101(101)	132(132)	132(132)
gen400_p0.9_75	**80**(79.16)	79(79)	114(114)	114(114)	136(136)	136(136)
keller6	63(63)	63(63)	93(91.24)	93(**91.68**)	108(105.32)	**117(114.56)**
p_hat1500-2	80(80)	80(80)	93(93)	93(93)	107(**107**)	107(106.92)
san400_0.7_2	**33(32.2)**	32(32)	47(**47**)	47(46.36)	61(61)	61(61)
san400_0.7_3	**28(27.4)**	27(27)	39(**38.52**)	39(38.08)	50(**50**)	50(49.76)
san400_0.9_1	102(102)	**103(102.2)**	150(150)	150(150)	200(200)	200(200)
frb50-23-1	67(65.84)	67(**66.24**)	79(78.12)	79(**78.96**)	92(90.12)	92(**91.16**)
frb50-23-2	66(65.64)	66(**66**)	79(78.08)	79(**78.96**)	91(89.8)	**92(90.72)**
frb50-23-3	65(63.44)	65(**64.04**)	76(75.12)	76(**75.68**)	87(86.16)	**88(87.4)**
frb50-23-4	66(65.16)	66(**65.88**)	78(77.56)	**79(78.96)**	91(89.6)	91(**90.72**)
frb50-23-5	67(65.48)	67(**66.24**)	79(78)	**80(79.16)**	91(89.84)	**92(91.12)**
frb53-24-1	70(69.2)	**71(70.88)**	84(82.96)	**85(84.12)**	97(96.32)	**99(97.96)**
frb53-24-2	69(68.4)	**70(69.96)**	82(80.96)	**83(82.16)**	94(93.28)	**95(94.6)**
frb53-24-3	70(68.56)	70(**69.84**)	83(81.72)	83(**82.84**)	95(94.2)	**96(96)**
frb53-24-4	**70**(68.4)	69(**69**)	82(81.32)	**84(82.52)**	95(94.12)	**96(95.68)**
frb53-24-5	68(67.44)	68(**68**)	80(79.72)	**82(81.52)**	92(91.84)	**94(93.52)**
frb56-25-1	74(73.24)	**75(74.28)**	89(87.6)	89(**88.96**)	102(101.52)	**104(103.04)**
frb56-25-2	74(72.96)	**75(74.48)**	88(86.8)	**89(88.48)**	101(100.12)	**103(102.04)**
frb56-25-3	73(72)	**74(73.48)**	87(85.4)	**88(87.04)**	99(98.44)	**101(100.04)**
frb56-25-4	73(72.08)	73(**72.84**)	87(85.04)	**88(86.88)**	99(98)	**100(99.72)**
frb56-25-5	73(72.2)	**74(73.084)**	86(85.4)	**88(86.8)**	99(98.32)	**101(100.08)**
frb59-26-1	78(76.64)	**79(78.16)**	92(90.72)	**93(92.64)**	106(104.72)	**108(106.96)**
frb59-26-2	78(76.88)	**79(78.16)**	91(90.96)	**93(92.8)**	106(104.8)	**108(106.8)**
frb59-26-3	77(75.6)	**78(77.04)**	91(89.92)	**93(92.2)**	105(104.08)	**107(106.12)**
frb59-26-4	77(75.68)	**78(77.2)**	91(89.84)	**93(92)**	105(103.6)	**107(105.84)**
frb59-26-5	77(76)	**78(77.92)**	91(89.56)	**92(91.68)**	105(103.24)	**106(105.48)**
frb100-40	125(123.4)	**129(127.6)**	148(146.64)	**156(153.68)**	170(168.72)	**178(176.56)**

DCCplex can return a larger solution than BnB on 24 instances most of the time.

DCCplex and FD-TS achieve the same quality solution quickly on most of these sparse graphs. To save space, we only report the results on the graphs that DCCplex and FD-TS do not return the same quality solutions in 100% runs with $k = 2, 3, 4$. As is shown in Table 2, DCCplex outperforms FD-TS on 14 instances in terms of average solution while it is defeated on only 5 instances, indicating its robustness. As for the best solution quality, DCCplex and FD-TS show the same performance in general.

Effectiveness of the DCC strategy

To study the effectiveness of the DCC strategy, we replace the DCC strategy in DCCplex with the SCC strategy. The resulting algorithm is termed as SCCplex. We compare DCCplex with SCCplex on the DIMACS and BHOSLIB benchmarks as well as massive graphs, and the result is summarized in Table 3.

We explain the meaning of each column in the table. We evaluate SCCplex and DCCplex on the three benchmarks with 19, 21 and 36 instances respectively. The column ”XCCplex #Better” represents the number of instances that XCCplex has better results than its competitor in terms of the maximum and average results. According to the Table 3, DCCplex outperforms SCCplex on DIMACS and BHOSLIB benchmarks with $k = 2, 3, 4$,. It is worth mentioning that DCCplex defeats SCCplex on 21, 20, 21 instances in terms of average results on BHOSLIB benchmark with $k = 2, 3, 4$ respectively while is defeated on one case. On those massive sparse graphs, the results of SCCplex and DCCplex are comparable. The experiment result verifies the effectiveness of the DCC strategy.

Empirical Analysis on Speedup Ratio

We assess the speedup ratio of DCCplex implemented with DINU scheme over DCCplex with straightfoward neighborhood updating scheme. The speedup ratio, denoted as r, is defined as

$$r = \frac{step(DINU)}{step(straightfoward)}$$

where the $step(DINU)$ (resp. $step(straightfoward)$) is the number of steps excuted by DCCplex with DINU (resp. straightfoward) scheme per 1000s. Note that we set the

Table 2: Evaluation on Massive Graphs with $k = 2, 3, 4$

Instance		FD-TS ct=100s	DCCplex ct=100s
rt-retweet-crawl	k=2	14(**13.5**)	14(13.3)
	k=3	15(**14.6**)	15(14.4)
	k=4	16(**15.6**)	16(15.5)
soc-digg	k=2	57(**55.8**)	57(55.3)
	k=3	63(61.8)	63(61.8)
	k=4	69(67.6)	69(**68.7**)
soc-FourSquare	k=2	35(34.5)	35(**34.9**)
	k=3	39(38.5)	39(**38.7**)
	k=4	42(38.3)	42(**42**)
soc-gowalla	k=2	30(**30**)	30(29.3)
	k=3	31(31)	31(31)
	k=4	32(32)	32(32)
soc-pokec	k=2	23(22.3)	**26(25.7)**
	k=3	**32**(28.1)	29(**28.8**)
	k=4	32(29.4)	32(**30.4**)
soc-twitter-follows	k=2	8(8)	8(8)
	k=3	9(8.9)	9(**9**)
	k=4	11(11)	11(11)
socfb-A-anon	k=2	28(26.2)	28(**27.3**)
	k=3	31(28.5)	**32(31.2)**
	k=4	34(32.3)	34(**34**)
socfb-B-anon	k=2	27(25.2)	27(**26.6**)
	k=3	30(28.4)	30(**29**)
	k=4	33(30.9)	33(**32**)

$\rho_0 = 1$ so that DINU always chooses the updating scheme of $O(k \cdot \Delta_{\overline{G}})$ complexity. Figure 2 is a scatter diagram that illustrates the speedup ratio varying with edge density of the graphs. Each point in the figure represents a graph in DIMACS/BHOSLIB. As we can see from Figure 2, it is a general trend that with the increase of edge density, the superiority of DCCplex with DINU scheme is increasingly obvious, which is in coincidence with the $O(k \cdot \Delta_{\overline{G}})$ complexity of DINU scheme. On 55 out of 60 instances, DCCplex with DINU scheme executes more iterations than its competitor within the same time. On 15 instances, the speedup ratio exceeds 10. It is worth mentioning that on the dense graph MANN_a81 whose edge density is 0.996, DCCplex with DINU scheme achieve the speedup ratio of 230. That explains why DCCplex with DINU scheme reaches the best-known value of the maximum 2-plex for MANN_a81 in 100% runs.

Conclusions and Future Work

In this paper, we have proposed a novel variant of Configuration Checking named DCC and the Double-attribute Neighborhood Update (DINU) scheme for the maximum k-plex problem. Based on the DCC strategy and DINU scheme, we develop a local search algorithm DCCplex. The experimental result shows that DCCplex achieve good performance across a broad range of problem instances and update the lower bounds on the size of the maximum k-plexes on many hard instances. In the future, we plan to study more clique relaxation models.

Table 3: Comparing SCCplex and DCCplex

Benchmarks		SCCplex #Better	DCCplex #Better
DIMACS(19)	k=2	1(1)	3(10)
	k=3	1(0)	1(9)
	k=4	0(0)	3(11)
BHOSLIB(21)	k=2	1(0)	5(21)
	k=3	0(1)	3(20)
	k=4	0(0)	7(21)
Massive(36)	k=2	0(0)	4(8)
	k=3	0(3)	0(1)
	k=4	0(0)	1(3)

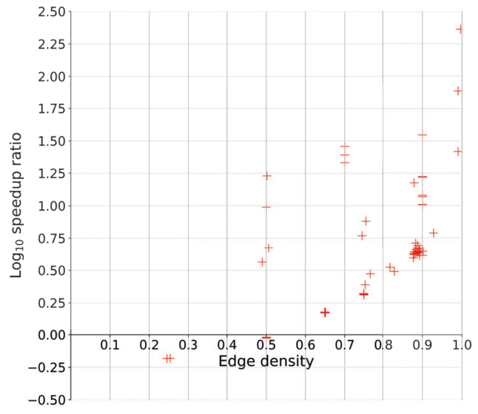

Figure 2: Speedup ratio varying with edge density

Acknowledgments

This paper was supported by the National Natural Science Foundation of China (No. 61573386, 61976232), National Key R&D Program of China (No. 2018YFC0830600), Guangdong Province Natural Science Foundation (No. 2016A030313292, 2017A070706010 (soft science), 2018A030313086), Guangdong Province Science and Technology Plan projects (No. 2016B030305007 and No. 2017B010110011), Guangzhou Science and Technology Project (No. 201804010435). Shaowei Cai was supported by Youth Innovation Promotion Association, Chinese Academy of Sciences (No.2017150).

References

Balasundaram, B.; Butenko, S.; and Hicks, I. V. 2011. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research* 59(1):133–142.

Battiti, R., and Protasi, M. 2001. Reactive local search for the maximum clique problem. *Algorithmica* 29(4):610–637.

Boginski, V.; Butenko, S.; and Pardalos, P. M. 2006. Mining market data: a network approach. *Computers & Operations Research* 33(11):3171–3184.

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., and Su, K. 2013. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence* 204:75–98.

Cai, S.; Lin, J.; and Luo, C. 2017. Finding A small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research* 59:463–494.

Cai, S.; Luo, C.; and Su, K. 2015. Improving walksat by effective tie-breaking and efficient implementation. *Computer Journal* 58(11):2864–2875.

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.

Conte, A.; De Matteis, T.; De Sensi, D.; Grossi, R.; Marino, A.; and Versari, L. 2018. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD*, 1272–1281.

Gao, J.; Chen, J.; Yin, M.; Chen, R.; and Wang, Y. 2018. An exact algorithm for maximum k-plexes in massive graphs. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 1449–1455.

Glover, F., and Laguna, M. 1998. Tabu search. In *Handbook of Combinatorial Optimization*. 2093–2229.

Gujjula, K. R.; Seshadrinathan, K. A.; and Meisami, A. 2014. A hybrid metaheuristic for the maximum k-plex problem. In *Examining Robustness and Vulnerability of Networked Systems*. 83–92.

Hoos, H. H., and Stützle, T. 2004. *Stochastic local search: Foundations and applications*. Elsevier.

Kondo, K., and Okubo, T. 2012. Structural estimation and interregional labour migration: Evidence from japan. Keio/Kyoto Joint Global COE Discussion Paper Series 2011-040.

Lakhlef, H. 2015. A multi-level clustering scheme based on cliques and clusters for wireless sensor networks. *Computers & Electrical Engineering* 48:436–450.

Lin, J.; Cai, S.; Luo, C.; Lin, Q.; and Zhang, H. 2019. Towards more efficient meta-heuristic algorithms for combinatorial test generation. In *Proceedings of Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering 2019*, 212–222.

Luce, R. D. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15(2):169–190.

Luo, C.; Yu, J.; Yu, D.; and Cheng, X. 2015. Distributed algorithms for maximum clique in wireless networks. In *Proceedings of the 11th International Conference on Mobile Ad-hoc and Sensor Networks*, 222–226.

Luo, C.; Su, K.; and Cai, S. 2012. Improving local search for random 3-sat using quantitative configuration checking. In *Proceedings of the 20th European Conference on Artificial Intelligence*, 570–575. IOS Press.

McClosky, B., and Hicks, I. V. 2012. Combinatorial algorithms for the maximum k-plex problem. *Journal of combinatorial optimization* 23(1):29–49.

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

Mokken, R. J. 1979. Cliques, clubs and clans. *Quality & Quantity* 13(2):161–173.

Pattillo, J.; Veremyev, A.; Butenko, S.; and Boginski, V. 2013. On the maximum quasi-clique problem. *Discrete Applied Mathematics* 161(1-2):244–257.

Pullan, W. 2007. Protein structure alignment using maximum cliques and local search. In *Proceedings of Advances in Artificial Intelligence 2007*, 776–780.

Rossi, R. A., and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 4292–4293.

Rossi, R. A.; Gleich, D. F.; Gebremedhin, A. H.; and Patwary, M. M. A. 2014. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International World Wide Web Conference*, 365–366.

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 *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 1514–1522.

Wang, Y.; Li, C.; Sun, H.; Chen, J.; and Yin, M. 2019. Mlqcc: an improved local search algorithm for the set k-covering problem. *International Transactions in Operational Research* 26(3):856–887.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 805–811.

Xiao, M.; Lin, W.; Dai, Y.; and Zeng, Y. 2017. A fast algorithm to compute maximum k-plexes in social network analysis. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 919–925.

Xu, K.; Boussemart, F.; Hemery, F.; and Lecoutre, C. 2005. A simple model to generate hard satisfiable instances. *arXiv preprint cs/0509032*.

Zhou, Y., and Hao, J.-K. 2017. Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research* 86:65–78.