*ECAI 2024*
*U. Endriss et al. (Eds.)*
*© 2024 The Authors.*
*This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).*
*doi:10.3233/FAIA240984*

# A Faster Branching Algorithm for the Maximum $k$-Defective Clique Problem

**Chunyu Luo**[a], **Yi Zhou**[a,*], **Zhengren Wang**[b] **and Mingyu Xiao**[a]

[a]University of Electronic Science and Technology of China, China
[b]Peking University, China
ORCID (Yi Zhou): https://orcid.org/0000-0002-9023-4374, ORCID (Mingyu Xiao): https://orcid.org/0000-0002-1012-2373

**Abstract.** A $k$-defective clique of an undirected graph $G$ is a subset of its vertices that induces a nearly complete graph with a maximum of $k$ missing edges. The maximum $k$-defective clique problem, which asks for the largest $k$-defective clique from the given graph, is important in many applications, such as social and biological network analysis. In the paper, we propose a new branching algorithm that takes advantage of the structural properties of the $k$-defective clique and uses the efficient maximum clique algorithm as a subroutine. As a result, the algorithm has a better asymptotic running time than the existing ones. We also investigate upper-bounding techniques and propose a new upper bound utilizing the *conflict relationship* between vertex pairs. Because the *conflict relationship* is common in many graph problems, we believe that this technique can be potentially generalized. Finally, experiments show that our algorithm outperforms state-of-the-art solvers on a wide range of open benchmarks. Our source code, as well as the experiment data, is open source and available here.

## 1 Introduction

Finding cohesive subgraphs is a fundamental task in many real-world network applications, such as community detection in social networks [2, 1, 13], protein complex mining in biological networks [31, 16], and statistical analysis in financial networks [6, 5]. A recent study in [22] also reveals a potential application of finding cohesive graphs in the design of Graph Neural Network [22]. The clique, which requires an edge for every pair of vertices, is a basic model for representing cohesive subgraphs. Additionally, the study of algorithms for the maximum clique problem, that is, finding the maximum clique from a given graph, has made significant progress both theoretically and practically in recent years. For example, the best-known time complexity for the exact computation of the maximum clique is $O^*(1.20^n)$ [29]; Practical algorithms such as MC-BRB [7] and MoMC [20] can solve massive sparse or hard dense graphs in seconds. However, the clique model itself is not a preferred choice for real-world applications. Due to the existence of noises and faults in many scenarios, the constraint that requires all possible relations to exist in the community may be too restrictive [11]. Therefore, various clique relaxations have been formulated in the literature, such as $k$-plex [26, 27], $k$-clique [3] and $k$-defective clique [31, 14, 8, 18].

This paper focuses on the $k$-defective clique model, a subset of vertices that allows the induced subgraph missing at most $k$ edges to be a complete subgraph. When $k$ is 0, the $k$-defective clique is a clique. For any $k \geq 0$, the $k$-defective clique problem is NP-hard [30] and W[1]-hard with respect to its solution size [19]. Furthermore, the problem cannot be approximated with a factor better than $O(n^\epsilon)$ for any $\epsilon > 0$ currently [17].

Although the maximum $k$-defective problem is difficult in theory, there have been several practical algorithms in the last decade, such as [25, 15, 10, 14, 8, 18]. All of these algorithms follow the branching algorithm scheme (also known as branch-and-bound, tree search, or backtracking search). Furthermore, [12] studied the enumeration of maximal $k$-defective cliques, which is also based on the branching algorithm.

From a complexity perspective, under the assumption that $k$ is a constant, branching algorithms have worst-case time complexity in the form of $O^*(\gamma_k^n)$ [1] where $n$ is the vertex number of the input graph and $\gamma_k$ is a constant related to $k$ but strictly smaller than 2 [10, 8]. Currently, the cutting-edge $\gamma_k$ is obtained by [8], for example $\gamma_1 = 1.84$ and $\gamma_2 = 1.93$.

From a practical perspective, the efficiency of branching algorithms is closely related to the tightness of upper bounds to prune unpromising branches. (That is why branching algorithms are also named branch-and-bound in the literature.) In particular, [10] and [14] proposed coloring and packing bounding rules, respectively, followed by [8] and [18] that improve these bounds. It should be mentioned that [28] studied the crown decomposition rule to reduce search space, but only from a theoretical point of view.

Despite a considerable number of existing algorithms, we noticed that some important properties of this problem are overlooked. Firstly, existing known branching algorithms are built from scratch, rather than making use of well-established algorithmic techniques like the maximum clique algorithm. In fact, there is a close relationship between a $k$-defective clique and a clique, that is, a $k$-defective clique is always a superset of a clique. It is possible to introduce the maximum clique algorithm for improving search efficiency. Second, let us say that two vertices conflict if they cannot be in the same maximum cohesive subgraph. Given two conflict vertices $u$ and $v$, one can reason with this information to derive a tight upper bound of the solution. As a result, there is a chance of further reducing the

* Corresponding Author. Email: zhou.yi@uestc.edu.cn

[1] The $O^*$ notation omits the polynomial factors.

search space of the algorithm. To our knowledge, pairwise *conflict relations* of the vertices are still under-explored. Based on the above observations, we make the following contributions in the paper.

1. **We design a new branching algorithm with a better worst-case runtime bound.** Our algorithm is based on the structural property that a $k$-defective clique can be partitioned into a so-called *$k$-defective set* and a clique. In a $k$-defective set, every vertex is not adjacent to at least one other vertex. The algorithm enumerates these $k$-defective sets and, for each such set $D$, calls the well-studied maximum clique algorithm to find the clique $C$ such that $D \cup C$ is a $k$-defective clique. When $k$ is a constant, the complexity of the algorithm is $O^*({\gamma_c}^n)$, where $\gamma_c$ is a maximum clique coefficient independent from $k$, e.g. $\gamma_c = 1.20$ by [29]. We notice that this simple approach leads to a better complexity than all known algorithms when $k$ is a constant. We further introduce a graph decomposition technique to reduce the runtime bound to $O^*({\gamma_c}^{d(G)})$, where $d(G)$ is a practically small parameter called graph degeneracy.
2. **We explore tighter upper bounds.** We introduce a novel method to effectively compute an upper bound for the problem. Specifically, we propose an upper bounding rule which incorporates the property of pairwise conflict vertices. (Two vertices conflict if they cannot be included in a sufficiently large cohesive subgraph. We also design a dynamic programming-based algorithm to approximate the upper bound following this rule. By both theoretical and practical analysis, we show that the bound obtained by our algorithm beats other state-of-the-art bounding techniques. To our knowledge, it is also the first time that a theoretical dominance relationship among different bounds has been established.

Extensive experiments demonstrate that our algorithm outperforms the state-of-the-art algorithms across three sets of large real-world graphs. We also analyze different bounds empirically, showing that the new bound can prune much more branches than the existing ones. A full version of the paper, including all the proofs and implementation details, is given in [21].

## 2 Preliminary

In an undirected graph $G = (V, E)$ where $V$ is the vertex set and $E$ is the edge set, $N_G(u)$ denotes the set of vertices adjacent to $u \in V$ in $G$ (when the context is clear, we use $N(u)$ instead), $G[S]$ denotes the subgraph induced by $S \subseteq V$, $CN(S)$ denotes the set of vertices in $G$ that are adjacent to every vertex in $S$, that is, $\bigcap_{u\in S} N(u)$[2]. The complement graph of $G$ is denoted as $\overline{G}$. Any two distinct vertices in $\overline{G}$ are adjacent if and only if they are not adjacent in $G$. Given a positive integer $n$, we use $[n]$ to denote the index set $\{1, 2, ..., n\}$.

A *clique* in $G = (V, E)$ is a set $S \subseteq V$ where all vertices in $S$ are pairwise adjacent in $G$. An *independent set* is a set $S \subseteq V$ where no two vertices in $S$ are adjacent in $G$. Given a non-negative integer $k$, a set $S \subseteq V$ is a $k$-defective clique if the subgraph $G[S]$ contains at least $\binom{|S|}{2} - k$ edges. Equivalently, $S$ is a $k$-defective clique in graph $G$ if the complement graph $\overline{G[S]}$ has at most $k$ edges. If $S$ is a $k$-defective clique in $G$, then any subset of $S$ is also a $k$-defective clique in $G$ [15]. For any subset $S \subseteq V$, we define $r(S) = k - |E(\overline{G[S]})|$ as the gap between $k$ and the number of edges in $\overline{G[S]}$. If $r(S) \geq 0$, then $S$ is a $k$-defective clique.

[2] $CN$ means "Common Neighbors"

**Lemma 1** (Two-hop Property [10]). *If $S$ is a $k$-defective clique in $G$ with $|S| \geq k + 2$, then $G[S]$ is a connected graph, and the length of the shortest path between any two vertices in $S$ is within 2.*

A $k$-defective clique with fewer than $k + 2$ vertices may be disconnected. For example, an isolated vertex and a $k$-clique form a $k$-defective clique of size $k + 1$. Because we are only interested in the connected and large $k$-defective cliques, we investigate the *non-trivial maximum $k$-defective clique problem* in the remaining.

**Definition 1** (Non-trivial Maximum $k$-Defective Clique Problem). *Given a graph $G$ and a non-negative integer $k$, find the largest $k$-defective clique $Q$ from $G$ such that its size is at least $k + 2$. If there is no such $k$-defective clique, return 'no'.*

In practice, almost all large real-world graphs have non-trivial maximum $k$-defective cliques, e.g. 113 out of 114 Facebook graphs [3] in the renowned network repository [24] have non-trivial $k$-defective clique when $k \leq 5$.

## 3 An Exact Decompose-and-Branch Algorithm

This section introduces a branching algorithm with a better exponential time bound than existing ones. We first introduce the notion of the $k$-defective set.

**Definition 2** ($k$-Defective Set). *A $D \subseteq V$ is a called a $k$-defective set of $G$ if $\forall u \in D$, $|N(v) \cap D| < |D| - 1$, and there are at least $\binom{|D|}{2} - k$ edges in $G[D]$.*

In general, a $k$-defective set $D$ is also a $k$-defective clique, except that every vertex in $D$ is nonadjacent to at least one other vertex in $D$. It is easy to see that the size of any $k$-defective set is no more than $2k$.

For any $k$-defective clique $Q$, we denote $D(Q)$ as the *maximal $k$-defective set* in $Q$, that is, we cannot find another vertex $v \in Q$ such that $D(Q) \cup \{v\}$ is still a $k$-defective set. Our algorithm is based on the observation that $Q$ can be partitioned into the maximal $k$-defective set $D(Q) \subseteq Q$ and a clique $C(Q) \subseteq Q$. We also observe that $C(Q) \subseteq CN(D(Q))$, that is, $\forall u \in D(Q)$, $C(Q) \subseteq N(u)$. If $D(Q)$ is empty, then $Q$ is a clique in $G$. Examples of this observation are given in Fig. 1.

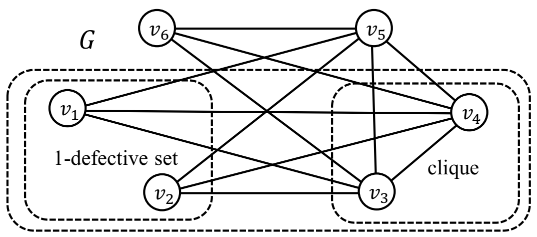


**Figure 1.** In graph $G$ of 6 vertices, for $S = \{v_1, v_2, v_3, v_4\}$, we have $G[S]$ is a 1-defective clique with $D(S) = \{v_1, v_2\}$ and $C(S) = \{v_3, v_4\}$.

### 3.1 The Branching Algorithm

By the above observation, our approach is outlined below. We enumerate all $k$-defective sets $D$ from $V$. For any non-empty $k$-defective set $D$, we compute the maximum clique $C^*$ within the subgraph induced by $CN(D)$. The maximum $k$-defective clique is the largest one among all candidates $D \cup C^*$.

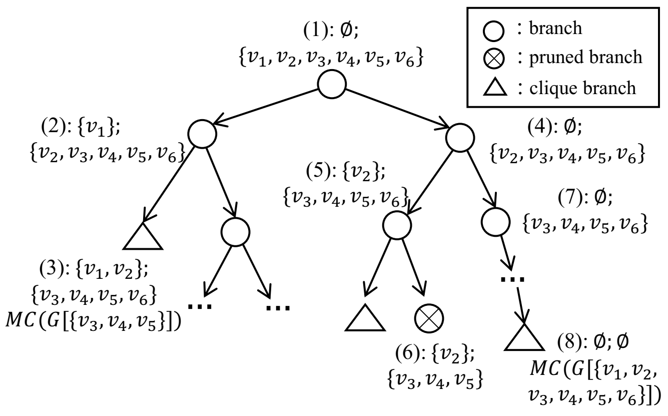
For clarity in the description of the algorithm, we define the *general $k$-defective clique problem.*

[3] https://networkrepository.com/socfb.php

**Definition 3** (General $k$-Defective Clique Problem)**.** *Given an instance $I = (G = (V, E), P, R)$ where $G$ is the graph, $P$ and $R$ are two disjoint subsets of $V$, find the maximum $k$-defective clique $Q^*$ such that $P \subseteq D(Q^*) \subseteq P \cup R$ and $Q^* \subseteq V$.*

We use $\omega_k(I)$ to denote the size of the maximum $k$-defective clique of instance $I$. When the context is clear, we use $n$ to represent the number of vertices in $G$.

In Alg. 1, we implement the above idea to solve the general $k$-defective clique problem with instance $I$. For any input $I$, we find a $k$-defective set $D$ such that $P \subseteq D \subseteq P \cup R$. For a non-empty set $D$, we find the maximum clique $C$ from the set $CN(D)$. If $D$ is empty, we find the maximum clique $C$ from $V$. Then, the maximum $k$-defective clique is the largest set among all $D \cup C$. The following example explains this tree search algorithm.



: branch
: pruned branch
: clique branch
(1): ∅; {v1, v2, v3, v4, v5, v6}
(2): {v1}; {v2, v3, v4, v5, v6}
(3): {v1, v2}; {v3, v4, v5, v6} MC(G[{v3, v4, v5}])
(4): ∅; {v2, v3, v4, v5, v6}
(5): {v2}; {v3, v4, v5, v6}
(6): {v2}; {v3, v4, v5}
(7): ∅; {v3, v4, v5, v6}
(8): ∅; ∅ MC(G[{v1, v2, v3, v4, v5, v6}])

**Figure 2.** The search tree of Alg. 1. Each node in the tree represents a branch. The two sets labeled after the node represents $P$ and $R$, respectively, when the branch is generated.

**Example** Assume that the graph $G$ in Fig. 1 is the input graph and $k = 1$. In Fig. 2, we present the branching tree of Alg. 1. Let us explain some representative branches.

- In branch (1), $P$ is initialized as $\emptyset$ and $R$ is $V$. By the branching rule at line 14 in Alg. 1, $v_1$ is selected as the branching vertex, and branches (2) and (4) are produced.
- In branch (2), by line 14 in Alg. 1, $v_2$ is the branching vertex, and two subbranches are produced.
- In branch (3), because $r(\{v_1, v_2\}) = 0$, by lines 6-9 in Alg. 1, then the $MC(\cdot)$ is called to find the maximum clique in $CN(P) = \{v_3, v_4, v_5\}$.
- In branch (6), $P = \{v_2\}$ and $v_2$ is adjacent to $v_3, v_4, v_5$. Because there is no $k$-defective set $D$ such that $\{v_2\} \subseteq D \subseteq \{v_2, v_3, v_4, v_5\}$, by lines 3-4 in Alg. 1, branch (6) is pruned.
- In branch (7), $\exists v_3 \in R$ that $v_3$ is all adjacent to $P \cup R$, by lines 11-12 in Alg. 1, $v_3$ is reduced in the subbranches.
- In branch (8), $R = \emptyset$, by line 6 in Alg. 1, $MC(\cdot)$ is called to find the maximum clique from $V$.

One can simply justify that the branching algorithm enumerates all $k$-defective sets that subsume $P$ from $P \cup R$ without repetition. Now we estimate the complexity of the branching algorithm.

**Lemma 2.** *Given an instance $I = (G, P, R)$ where $P \neq \emptyset$, the branching algorithm finds the maximum $k$-defective clique of $I$ in time $O(|R|^{2k}\gamma_c^m)$ where $m = |CN(P)|$ and $\gamma_c$ is the base factor in the time complexity for maximum clique algorithm.*

**Algorithm 1:** The branching algorithm for the maximum $k$-defective clique problem based on $k$-defective set enumeration.

```
   Input: An instance $I = (G, P, R)$ and and a known solution $Q^*$
   Output: A maximum $k$-defective clique $Q^*$ of $I$
1  Procedure branching($I = (G = (V, E), P, R)$)
2  begin
3      if $\exists u \in P$ that $u$ is adjacent to all vertices in $P \cup R$ or $r(P) < 0$ then
           /* $P$ is not included in any $k$-defective set. */
4          return
5      if $r(P) = 0$ or $R = \emptyset$ then
           /* $P$ is a $k$-defective set and no more branching */
6          Let $C$ be $CN(P)$ if $P$ is non-empty, otherwise $V$
7          $Q \leftarrow$ MC($G[C]$) $\cup P$
8          if $|Q| > |Q^*|$ then
9              $Q^* \leftarrow Q$
10         return
11     if $\exists v \in R$ that $u$ is adjacent to all vertices in $P \cup R$ then
12         branching($I' = (G, P, R \setminus \{v\})$)
13     else
14         Find a branching vertex $v$ in $R$ such that $v = \arg\max_{u \in R}(|P| - |P \cap N(u)|)$
15         branching($I'_1 = (G, P \cup \{v\}, R \setminus \{v\})$)
16         branching($I'_2 = (G, P, R \setminus \{v\})$)
```

*Proof Sketch.* Because the size of a $k$-defective set is bounded by $2k$, there are at most $|R|^{2k}$ different $k$-defective sets. For each $k$-defective set $D$ that $D \supseteq P$, we find the maximum clique algorithm in $CN(D) \subseteq CN(P)$ with a time complexity of $O^*(\gamma_c^{|CN(P)|})$. So, the complexity of the branching algorithm is $O^*(|R|^{2k}\gamma_c^{|CN(P)|})$. □

To our knowledge, $\gamma_c$ can be as small as 1.20 by [29]. In our experiments, we use the maximum clique algorithm in [7]. As far as we know, this is one of the most practically efficient maximum clique algorithms in large real-world graphs, despite there being no analysis of its time complexity.

## 3.2 The Whole Decompose-and-Branch Algorithm

We further investigate the *graph decomposition rule* in the algorithm. The rule reduces the original input instance $I = (G, \emptyset, V)$ to instances $I_1, ..., I_n$ such that the solution of $I$ is the maximum size in these $\omega_k(I_i)$.

Because the rule is defined with an order of set $V$, let us first denote the order as $v_1, v_2, ..., v_n$. We also denote $N(v_i) \cap \{v_{i+1}, ..., v_n\}$ as $N^+(v_i)$, and $\left(\bigcup_{v \in N^+(v_i)} N(v)\right) \cap \{v_{i+1}, ..., v_n\} \setminus N^+(v_i)$ as $N^{2+}(v_i)$. ($N^{2+}(v_i)$ is a subset of vertices that rank after $v_i$, and each $u \in N^{2+}(v_i)$ is not in $N^+(v_i)$ but adjacent to at least one vertex in $N^+(v_i)$). We say that $N^{2+}(v_i)$ is two-hop adjacent to $v_i$ with respect to the given order.

**Lemma 3.** *Give an instance $I = (G, \emptyset, V)$ and an order of $V$ which is denoted by $v_1, ..., v_n$. Then $\omega_k(I) = \max_{i=[n]} (\omega_k(I'_i), \omega_k(I''_i))$*

*where $I_i' = (G[\{v_i\} \cup N^+(v_i) \cup N^{2+}(v_i)], \{v_i\}, N^+(v_i) \cup N^{2+}(v_i))$ and $I_i'' = (G[\{v_i\} \cup N^+(v_i)], \emptyset, N^+(v_i))$.*

To understand this lemma, one can think that in a non-trivial $k$-defective clique, the length of the shortest path of any two vertices is bounded by 2. So we can find the largest $k$-defective clique from all diameter-two bounded subgraphs.

Clearly, a order with small $|N^+(v_i)|$ and $|N^{2+}(v_i)|$ is preferred. We introduce the *degeneracy ordering*. Given a graph $G$ with $n$ vertices, the degeneracy ordering is an order $v_1, v_2, ..., v_n$ such that the largest $|N^+(v_i)|$ for all $i \in [n]$ is minimized. A degeneracy ordering of $G = (V, E)$ can be computed in time $O(|V| + |E|)$ by repeatedly removing a vertex with the minimum degree in the remaining graph until the graph becomes empty [23]. Given a degeneracy ordering, $|N^+(v_i)|$ is bounded by the *degeneracy of* $G$ which is normally denoted as $d(G)$. Note that the degeneracy ordering has also been investigated in many maximum clique and near-clique problems [7, 32, 27].

Now, we integrate the decomposition rule with our branching algorithm and obtain the whole decompose-and-branch algorithm DnBk in Alg. 2. In line 3, we use a linear-time heuristic algorithm like that in [8] to find an initial $k$-defective clique (see the supplementary for a complete description of this heuristic). This initial solution is used to prune branches using our upper bounding rule in the following section. In lines 7 and 9, we call the branching algorithm with instances $I_i'$ and $I_i''$, respectively.

**Algorithm 2:** The decomposition-and-branching algorithm for maximum $k$-defective clique

```
   Input: A graph $G$ and an positive integer $k$, where $n = |V|$
   Output: A maximum $k$-defective clique $Q^*$ in $G$
1  Procedure DnBk($G, k$)
2  begin
3      Use a linear-time heuristic to find an initial solution $Q$
         from $G$
4      Order $V$ as $v_1, ..., v_n$ by degeneracy ordering
5      for $i \in [n]$ do
6          Build $I_i' = (G[\{v_i\} \cup N^+(v_i) \cup$
             $N^{2+}(v_i)], \{v_i\}, N^+(v_i) \cup N^{2+}(v_i))$
7          branching($I_i'$)
8          Build $I_i'' = (G[\{v_i\} \cup N^+(v_i)], \emptyset, N^+(v_i))$
9          branching($I_i''$)
10     return $Q^*$
```

**Theorem 4.** *Given a graph G and a constant non-negative integer k, denoting the degeneracy of G as d(G), the DnBk algorithm finds the maximum k-defective clique in time $O^*({\gamma_c}^{d(G)})$, where $\gamma_c$ is the base of the exponential factor in the time complexity for maximum clique algorithm.*

*Proof.* First, let us consider instance $I_i'$ where $P = \{v_i\}$ and $R = \{N^+(v_i) \cup N_G^{2+}(v_i)\}$. Clearly, we have $|R| \leq \min(d(G)\Delta(G) + d(G), |V|)$. On the other hand, because $v_i \in P$, so $CN(P)$ is bounded by $|N^+(v_i)|$ and $|N^+(v_i)| \leq d(G)$. By Lemma 2, the time complexity of the branching algorithm with $I_i'$ is $O((d(G)\Delta + d(G))^{2k+O(1)}\gamma_c^{d(G)})$. Second, consider instance $I_i''$ where $G = G[\{v_i\} \cup N_G^+(v_i)]$, $P = \emptyset$ and $R = N^+(v_i)$. Clearly, we have $|R| \leq d(G)$ and $|CN(P)| \leq |\{v_i\} \cup N_G^+(v_i)| \leq d(G) + 1$. Consequently, the time complexity of the branching algorithm with $I_i''$ is $O(d(G)^{2k+O(1)}\gamma_c^{d(G)+1})$. Because $k$ is a constant, the total time complexity of DnBk is $O^*(\gamma_c^{d(G)})$. □

**Remark** Under the assumption that $k \geq 1$ is a constant, this time complexity improves the known results from two perspectives. First, the exponential base $\gamma_c$, which can be as small as 1.2 so far, is better than the state-of-the-art [9]. Second, the parameter (exponent) $d(G)$ is also smaller than $n$ in large graphs. For example, in soc-Livejournal, $d(G)$ is 213 in contrast to $n = 4033137$.

## 4 Exploring Tighter Upper Bound

We investigate the *upper bounding rule* to prune the branching algorithm. Given an instance $I = (G = (V, E), P, R)$, the bounding rule introduces an upper bound on $\omega_k(I)$. Suppose that there is currently a best-known lower bound, say $lb$. (Initially, $lb = |Q|$ where $Q$ is found using a linear-time heuristic.) In the branching algorithm, if the upper bound of $I$ is not greater than $lb$, we can claim that there is no hope of finding a $k$-defective clique larger than $lb$ in $I$, and thus we can stop the search. This is the principle of bounding in branch-and-bound algorithms.

### *4.1 Revisiting the Existing Upper Bounds*

We first review the *packing bound* and the *coloring bound* rules that are independently proposed in [14] and [10].

Before that, in an instance $I = (G, P, R)$, we define the *weight of* $u \in V$ as $w(u) = |P| - |N(u) \cap P|$. The $w(u)$ can be seen as the number of vertices that are not adjacent to $u$ in $P$. Clearly, when moving some vertices from $V \setminus P$ to $P$, the sum of weight cannot exceed $r(P)$, otherwise $P$ is not a $k$-defective clique anymore. This is the essence of packing bound.

**Bounding Rule 1** (Packing Bound [14])**.** *Assume that the vertices in $V \setminus P$ are sorted in non-decreasing order of weight $w(v)$, say $v_1, ...., v_{|V \setminus P|}$. Then, $|P| + i$ is an upper bound of $\omega_k(I)$ where $i$ is the largest number that $\sum_{j=1}^{i} w(v_j) \leq k$.*

The coloring bound depends on a graph coloring partition of $V \setminus P$. That is to say, $V \setminus P$ should be partitioned into $\chi$ non-disjoint independent sets before computing the coloring bound.

**Bounding Rule 2** (Coloring Bound [10])**.** *Assume $V \setminus P$ is partitioned into $\chi$ non-disjoint independent sets $\Pi_1, ..., \Pi_\chi$. Then $|P| + \sum_{j=1}^{\chi} min(\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor, |\Pi_j|)$ is an upper bound of $\omega_k(I)$.*

This bound holds because at most $\lfloor \frac{1+\sqrt{8k+1}}{2} \rfloor$ vertices in an independent set can be contained in the same $k$-defective clique.

Recently, a *Sorting* bound was proposed in [8], and another partition-based bound called *Club* was given in [18]. These bounding rules are more complicated than the packing and coloring bounds. In the following, we will investigate the *conflict relationship* of the vertices and design a new bounding framework that hybrids the packing, coloring bounds and the conflict relationship.

### *4.2 A New Upper Bound Based on Packing, Coloring and Conflict Vertices*

Let us first introduce *conflict vertices*.

**Definition 4** (Conflict Vertices). *Give an instance $I = (G, P, R)$ and a lower bound lb, two different vertices $\{u, v\}$ $(u, v \in V)$ are conflict vertices if $u$ and $v$ cannot be in the same k-defective clique of size larger than lb.*

There are several rules to identify the conflict vertices for a given instance. So far, we have used the following rules.

1. If $u \in R$, $v \in V \setminus (P \cup R)$ and $\{u, v\} \notin E$, then $\{u, v\}$ are conflict vertices.
2. If $u, v \in V \setminus (P \cup R)$ and $\{u, v\} \notin E$, then $\{u, v\}$ are conflict vertices.
3. If $u, v \in V \setminus P$ and $r(P \cup \{u, v\}) < 0$, then $\{u, v\}$ are conflict vertices.
4. If $u, v \in V \setminus P$, $\{u, v\} \in E$ and $|N(u) \cap N(v) \cap V \setminus P| \leq lb - (|P| + r(P) - w(u) - w(v) + 2)$, then $\{u, v\}$ are conflict vertices.
5. If $u, v \in V \setminus P$, $\{u, v\} \notin E$ and $|N(u) \cap N(v) \cap V \setminus P| \leq lb - (|P| + r(P) - w(u) - w(v) + 1)$, then $\{u, v\}$ are conflict vertices.

For each pair of vertices $u, v \in V \setminus P$, we check whether $\{u, v\}$ are conflict vertices. In each branch, we keep updating the conflict information. Note that more rules can be possibly discovered for future extensions. We use these rules because they are easy to implement and identifying them does not require too much computation effort.

In the following, we denote $conflict(u, v) = 1$ if $u$ and $v$ are conflict vertices and $conflict(u, v) = 0$ otherwise. Now, we are ready to formulate our bounding rule with this conflict information, namely, the *packing, coloring with conflict bounding rules*.

**Bounding Rule 3** (The Packing, Coloring with Conflict Bounding Rules). *Given an instance $I = (G, P, R)$ and a lower bound lb, assume that $V \setminus P$ is partitioned into $\chi$ independent sets $\Pi_1, ..., \Pi_\chi$. For any $u, v \in V \setminus P$, $conflict(u, v) = 1$ if $\{u, v\}$ are conflict vertices and $conflict(u, v) = 0$ otherwise.*

*The optimal objective value of the optimization problem described in the following is an upper bound of $\omega_k(I)$.*

$$\max_{S_i \subseteq \Pi_i, \forall i \in [\chi]} \quad |P| + \sum_{i=1}^{\chi} |S_i| \qquad \textbf{\textit{OPT}}$$

$$\text{s.t.} \sum_{i=1}^{\chi} \left( \binom{|S_i|}{2} + \sum_{u \in S_i} w(u) \right) \leq r(P) \qquad (1)$$

$$conflict(u, v) = 0, \forall u, v \in S_i \ \forall i \in [\chi] \qquad (2)$$

The optimization problem OPT asks for a subset $S_i$ from each $\Pi_i$ such that $S_i$ meets the (1), the *packing-and-coloring constraint* and (2), the *conflict constraint*. When $\chi = 1$, the OPT problem is a special case of the NP-hard knapsack problem with conflict graph [4]. Hence, it is unlikely to efficiently solve the optimization problem due to its NP-hard nature.

However, with the conflict constraint partially satisfied, we can get a relaxed OPT problem and the optimal solution to the relaxed OPT problem is still a feasible bound of $\omega_k(I)$. In the following, we first introduce an algorithm to solve the relaxed OPT problem which excludes the conflict constraint (2). Then, we improve this algorithm so that some of the conflict constraint 2 can be satisfied. Clearly, the objective value is an upper bound of the OPT problem, which is in turn, an upper bound of $\omega_k(I)$.

### 4.2.1 Dynamic Programming without Conflict Constraint

We propose the following dynamic programming-like algorithm to solve the OPT without considering the conflict constraint.

**DPBound Algorithm**

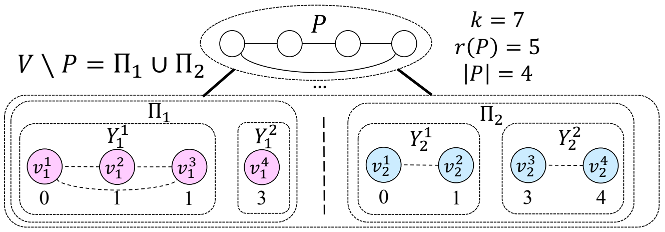
1. For each independent set $\Pi_i$ where $i \in [\chi]$, order the vertices in $\Pi_i$ by the non-decreasing order of the weight $w(\cdot)$. Assume that $\Pi_i$ is ordered as $v_i^1, v_i^2, ..., v_i^{|\Pi_i|}$. For each $r \in \{0, ..., r(P)\}$, find the maximum number $j \in \{0, ..., |\Pi_i|\}$ such that $\binom{j}{2} + \sum_{k=1}^{j} w(v_i^k) \leq r$, and assign $t(i, r) = j$.
2. For each $i \in [\chi]$ and $r \in \{0, ..., r(P)\}$, define $f(i, r)$ as the maximum value of $\sum_{j=1}^{i} |S_j|$ such that $\sum_{j=1}^{i} \left( \binom{|S_j|}{2} + \sum_{u \in S_j} w(u) \right) \leq r$ and $S_j \subseteq \Pi_j$ $(\forall j \in [i])$. Then $f(i, r)$ can be computed by the following recurrence relations.

$$f(1, r) = t(1, r), \quad \forall r \in \{0, ..., r(P)\}$$
$$f(i, r) = \max_{s \in \{0, ..., r\}} f(i-1, s) + t(i, r-s),$$
$$\forall i \in \{2, ..., \chi\} \ \forall r \in \{0, ..., r(P)\}$$

3. Return $|P| + f(\chi, r(P))$ as the optimal objective value.

For the DPBound algorithm, the first step can be easily implemented in time $O(|V \setminus P|)$ using a linear-time sorting algorithm. In the second step, we employ conventional bottom-up dynamic programming to resolve the recurrence relation, which has running time $O(\chi \cdot r(P)^2)$. The total running time is $O(|V \setminus P| + \chi \cdot r(P)^2)$.

**Example** We show an example of the DPBound algorithm in Fig. 3. In the example, $V \setminus P$ is partitioned into two independent sets $\Pi_1 = \{v_1^1, v_1^2, v_1^3, v_1^4\}$ and $\Pi_2 = \{v_2^1, v_2^2, v_2^3, v_2^4\}$, and each set is sorted in non-decreasing order of the weight $w(\cdot)$. The two tables on the left of the bottom show the results of $t(i, r)$ and $f(i, r)$ by running the DPBound algorithm. Finally, the bound is $|P| + f(2, 5) = 8$.



$t(i,r)$

| $r \backslash i$ | 1 | 2 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 2 | 2 |
| 4 | 2 | 2 |
| 5 | 3 | 2 |

$f(i,r)$

| $r \backslash i$ | 1 | 2 |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 2 | 3 |
| 4 | 2 | 4 |
| 5 | 3 | 4 |

Computation of DPBound

$t(i,r)$

| $r \backslash i$ | 1 | 2 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 2 | **1** | **1** |
| 3 | **1** | **1** |
| 4 | 2 | 2 |
| 5 | **2** | 2 |

$f(i,r)$

| $r \backslash i$ | 1 | 2 |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 1 | 2 |
| 2 | **1** | **2** |
| 3 | **1** | **2** |
| 4 | 2 | **3** |
| 5 | **2** | **3** |

Computation of PackColorConf

**Figure 3.** An example of running the DPBound with and without conflict vertices. The dashed lines represent conflict relations. The weight of vertices in $V \setminus P$, $w(\cdot)$, is displayed below the vertices.

### 4.2.2 The Algorithm for Packing, Coloring and Conflict Bound

Now, we propose an algorithm to solve OPT such that the packing-and-coloring constraint is satisfied and the conflict constraint is partially satisfied.

**Algorithm 3:** Approximate the packing, coloring bound with conflict constraint.

```
Input: An instance $I = (G, P, R)$, a known solution $Q'$, and
       a partition of $V \setminus P$ into independent sets
       $\mathbf{\Pi} = \{\Pi_1, ..., \Pi_\chi\}$.
Output: An upper bound of $\omega_k(I)$.
1  PackColorConf($I = (G, P, R), lb, \mathbf{\Pi}$)
2  begin
3      Build up the conflict function $conflict(\cdot)$ for each pair of
         vertices $\{u, v\}$ in $V \setminus P$
4      for $\Pi_i$ from $\Pi_1$ to $\Pi_\chi$ do
5          $\sigma \leftarrow 1, \mathcal{Y} \leftarrow \{\emptyset\}$
6          while $\Pi_i \neq \emptyset$ do
7              Sort $\Pi_i$ by non-decreasing order of $w(\cdot)$
8              Build an empty set $Y_\sigma$
9              for each $v$ in $\Pi_i$ by the sorted order do
10                 if $Y_\sigma = \emptyset$, or $\forall u \in Y_\sigma$ $conflict(u, v) = 1$
                   then
11                     $Y_\sigma \leftarrow Y_\sigma \cup \{v\}$
12             $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{Y_\sigma\}, \Pi_i \leftarrow \Pi_i \setminus Y_\sigma$
13             $\sigma \leftarrow \sigma + 1$
14         for $r$ from 0 to $r(P)$ do
15             Compute the maximum index $j \in [\sigma]$ such that
               $\binom{j}{2} + \sum_{l=1}^{j} (\min_{u \in Y_l} w(u)) \leq r$.
16             $t(i, r) \leftarrow j$
17     Use the linear recurrence in step 2 in the DPBound
         algorithm and compute $f(\chi, r(P))$
18     return $|P| + f(\chi, r(P))$
```

The general idea of PackColorConf is as follows. We partition each $\Pi_i$ into subsets $Y_1, ..., Y_\sigma$ such that vertices in each set $Y_j$ ($j \in [\sigma]$) are mutually conflicted (lines 6-13). Then, for each $\Pi_i$, we compute the values of $t(i, r)$ for each $r \in \{0, ..., r(P)\}$ by line 15. If the partition of $\Pi_i$ is non-trivial (that is, a subset $Y_j$ has more than one vertex), we may obtain $t(i, r)$ values smaller than in the first step of PackColorConf, thus tightening the final bound.

**Example** We show an example of Alg. 3 in Fig. 3. Suppose that $\Pi_1$ is partitioned into two sets $Y_1^1$ and $Y_1^2$ such that vertices in each set are conflicted to each other. Taking the computation of $t(1, 3)$ as an example. Because $\arg\min_{u \in Y_1^1} w(u) = v_1^1, \arg\min_{u \in Y_1^2} w(u) = v_1^4$ and $w(v_1^1) + w(v_1^4) + 1 > 3$ but $w(v_1^1) + 0 < 3$, we can get $t(1, 3) = 1$ which is smaller than that 2 obtained in the DPBound algorithm. After computing all the values of $t(i, r)$, we calculate the final bound $|P| + f(2, 5) = 7$. This is smaller than 8, the bound obtained by the DPBound algorithm.

### 4.3 Comparison of Bounding Rules

We compare the recent bounding rules for the maximum $k$-defective clique through the dominance relations. For a maximizing problem, we say that *Bounding Rule A dominates Bounding Rule B* if, for the same instance, the bound found by A is not larger than that found by B. In Fig. 4, we show the dominance relations among the packing bound [14], coloring bound [10], sorting bound [8], club bound [18] and the bound obtained by PackColorConf. Note that PackColorConf only obtains the upper bound of the OPT problem (Section 4.2.2). We can see that PackColorConf dominates the Sorting bound, which in turn dominates the Coloring and the Packing bounds. Due to the heuristic nature of the Club, we only know that it dominates Packing so far.

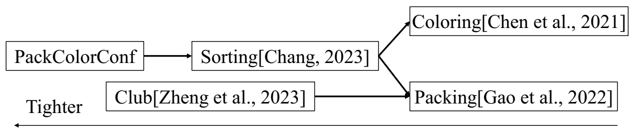


**Figure 4.** The dominance relations among bounding rules. $A \longrightarrow B$ indicates that A dominates B.

## 5 Experiments

In this section, we evaluate our algorithms and bounds empirically. Our algorithm is written in C++11 and compiled by g++ version 9.3.0 with the -Ofast flag. All experiments are conducted on a machine with an Intel(R) Xeon(R) Gold 6130 CPU @ 2.1GHz and a Ubuntu 22.04 operating system. Hyper-threading and turbo are disabled for steady clock frequency.

We empirically compare DnBk with the state-of-the-art algorithms kDC [8], KDClub [18]. Three real-world datasets are used as benchmark graphs, the same as in [8, 14]. Because the kDC-2 codes [9] are not publicly available, we only partially compare this algorithm using the data in the paper. This is reported in Section 6.2 in the supplementary file.

- **The NDR** dataset contains 139 real-world graphs with up to $5.87 \times 10^7$ vertices from the Network Data Repository [4], including social networks, biological networks, collaboration networks and so on.
- **The DIMACS10&SNAP** dataset contains 37 graphs with up to $5.61 \times 10^6$ vertices, 27 of them are from the DIMACS10 competition[5], and 10 are from SNAP library [6].
- **The Socfb** set contains 114 graphs with up to $1.18 \times 10^8$ vertices, extracted from Facebook social networks. [7]

We record the computation time for running an algorithm on a graph instance for a specific $k$ from $\{1, 5, 10, 15, 20\}$, with a cutoff time 10800 seconds (3 hours) for each test. This configuration is the same as that in [8]. The recorded time excludes the I/O time of loading the graph instance from the disk to the main memory. Because the problem we solve is to find the maximum $k$-defective clique of size larger than or equal to $k+2$, we omit instances whose maximum $k$-defective clique size is smaller than $k+2$ for each $k$.

### 5.1 Overall Performance

In Fig. 5, we show the number of solved instances within different time frames. The codes for KDBB [14] have not been open source. So, it is not included in the figure. In fact, both kDC and KDClub outperform KDBB in their reports. Our DnBk also solves all instances that were solved in [14] but consumes much less time.

From Fig. 5, DnBk solves more instances or at least as many instances as kDC and KDClub in 3 hours for all sets. For small $k$ values such as 1 or 5, these algorithms can solve almost all instances of DIMACS10&SNAP and Socfb to optimal in 3 hours, but DnBk still solves more in a shorter time frame. For $k = 10, 15$ and 20, DnBk clearly outperforms kDC and KDClub.

[4] https://networkrepository.com/
[5] https://networkrepository.com/dimacs10.php
[6] http://snap.stanford.edu/data/
[7] https://networkrepository.com/socfb.php

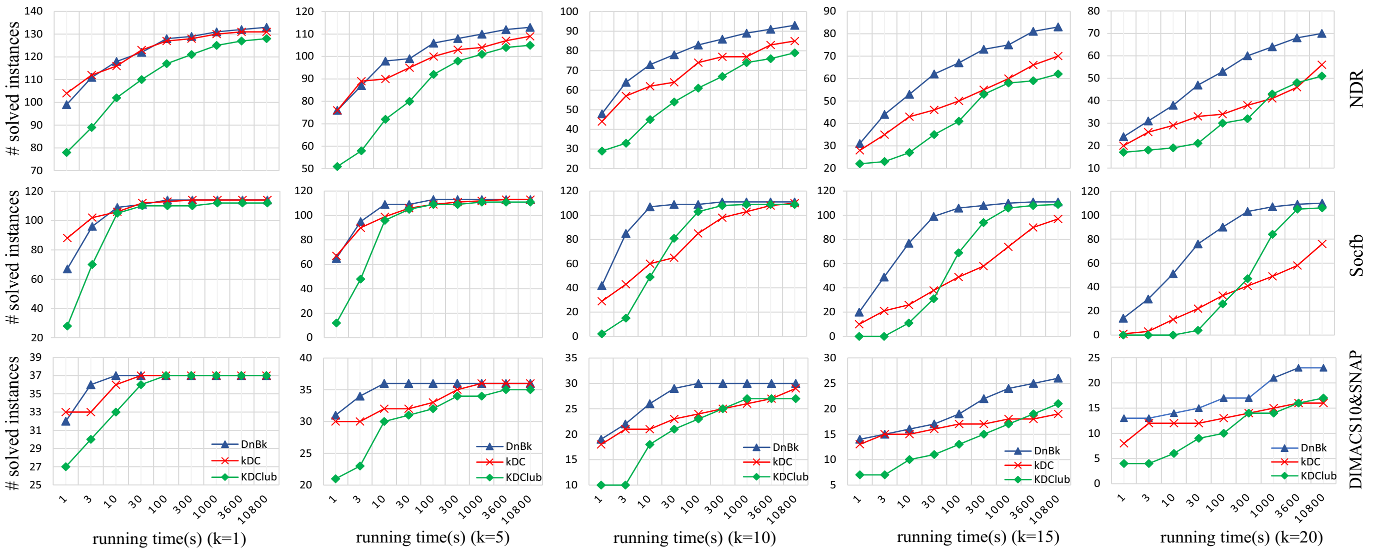


**Figure 5.** Number of solved instances for NDR, Socfb and DIMACS10&SNAP graphs, with $k = 1, 5, 10, 15, 20$ and a cutoff time 10800s.

Specifically, we find that DnBk closes 3,2,5,8 and 6 NDR instances that were not solved by other algorithms for $k = 1, 5, 10, 15$ and 20, respectively.

**Table 1.** The time (in seconds) and search tree nodes (branch) number (in $10^3$ scale) of different algorithms on example graphs. We show cases where $k = 1, 10, 20$. *opt* indicates the maximum solution size. *OOT* means a failure due to out-of-time. The instances that cannot be solved by any algorithms within 10800s are omitted.

| Graph | k | opt | #Nodes($10^3$) | | | | | Time(s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | DnBk | DnBk$_S$ | DnBk$_{Club}$ | DnBk$_P$ | DnBk$_C$ | DnBk | DnBk$_S$ | DnBk$_{Club}$ | DnBk$_P$ | DnBk$_C$ |
| socfb-Auburn71 | 1 | 58 | **0.1** | 0.1 | 0.1 | 3.1 | 0.2 | 3.3 | **3.2** | 3.9 | 4.6 | 6.6 |
| | 10 | 63 | **14.2** | 94.4 | 92.1 | 4902.4 | 1084.3 | **6.8** | 21.2 | 21.8 | 2389.0 | 712.2 |
| | 20 | 67 | **1274.8** | 7720.3 | 10349.2 | OOT | OOT | **233.5** | 1329.9 | 1801.3 | OOT | OOT |
| socfb-Tennessee | 1 | 59 | **0.0** | 0.0 | 0.0 | 0.6 | 0.0 | 2.1 | **2.0** | 2.3 | 3.3 | 3.9 |
| | 10 | 66 | **0.6** | 16.4 | 16.5 | 1339.0 | 29.8 | **2.1** | 4.1 | 4.3 | 418.4 | 17.0 |
| | 20 | 70 | **125.5** | 2045.0 | 2590.3 | 9870.6 | 4720.0 | **26.0** | 406.7 | 395.0 | 3395.2 | 789.5 |
| socfb-Texas84 | 1 | 52 | 0.2 | 0.2 | **0.1** | 6.4 | 0.3 | **4.1** | 4.5 | 5.2 | 5.9 | 8.0 |
| | 10 | 58 | **8.4** | 40.0 | 38.0 | 4730.7 | 2230.3 | **6.2** | 11.9 | 14.5 | 2379.0 | 1641.0 |
| | 20 | 61 | **3193.5** | 17647.5 | 22635.2 | OOT | OOT | **543.2** | 3182.7 | 3453.5 | OOT | OOT |
| tech-WHOIS | 1 | 59 | 0.2 | 0.2 | **0.1** | 5.4 | 0.3 | 0.5 | **0.5** | 0.6 | 1.8 | 1.3 |
| | 10 | 64 | **95.7** | 849.5 | 836.0 | OOT | 3367.5 | **22.5** | 173.9 | 147.3 | OOT | 3439.0 |
| | 20 | 69 | **5667.7** | OOT | OOT | OOT | OOT | **1265.1** | OOT | OOT | OOT | OOT |
| soc-Epinions1 | 1 | 24 | 0.4 | 0.4 | **0.3** | 5.5 | 1.0 | 5.2 | 5.2 | 7.1 | **4.6** | 10.3 |
| | 10 | 29 | **131.7** | 507.2 | 511.0 | 13029.3 | OOT | **22.2** | 51.6 | 54.3 | 3199.5 | OOT |
| | 20 | 32 | **26184.3** | 41184.5 | 97583.9 | OOT | OOT | **2072.6** | 3433.7 | 5738.3 | OOT | OOT |
| email-EuAll | 1 | 17 | **0.0** | 0.0 | 0.0 | 0.1 | 0.0 | **0.2** | 0.2 | 0.3 | 0.2 | 0.4 |
| | 10 | 21 | **39.0** | 61.8 | 80.4 | 1229.5 | 3375.9 | **4.1** | 5.0 | 6.2 | 117.1 | 371.9 |
| | 20 | 24 | **1912.4** | 2034.8 | 9827.5 | OOT | OOT | **360.4** | 446.5 | 776.9 | OOT | OOT |
| as-22july06 | 1 | 18 | **0.0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | **0.0** | **0.0** | 0.0 | 0.0 |
| | 10 | 22 | **2.4** | 2.6 | 5.0 | 18.7 | 24.9 | **0.3** | 0.3 | 0.6 | 2.8 | 3.5 |
| | 20 | 24 | **474.7** | 483.5 | 1591.8 | 14409.8 | OOT | **47.9** | 61.9 | 115.5 | 1630.6 | OOT |
| soc-orkut | 1 | 48 | 0.5 | 0.5 | **0.4** | 261.5 | 1.9 | **718.7** | 768.7 | 782.2 | 1250.9 | 1026.8 |
| | 10 | 53 | **157.3** | 422.7 | 313.5 | OOT | OOT | **936.8** | 1360.3 | 1324.8 | OOT | OOT |
| rt-retweet-crawl | 1 | 14 | **0.0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.7 | **0.6** | 0.6 | 1.5 | 1.1 |
| | 10 | 17 | **0.0** | 0.0 | 0.8 | 4.6 | 5.9 | 0.9 | **0.7** | 0.9 | 1.8 | 2.1 |
| tech-as-skitter | 1 | 68 | **0.0** | 0.0 | 0.0 | 0.0 | 0.0 | 1.9 | 1.8 | 1.9 | **1.7** | 3.3 |
| | 10 | 72 | **1.1** | 1.2 | 1.8 | 13.6 | 9.3 | **4.0** | 4.4 | 4.7 | 26.2 | 15.6 |
| | 20 | 75 | **24.0** | 24.8 | 54.5 | OOT | OOT | **20.0** | 20.9 | 32.9 | OOT | OOT |
| soc-youtube | 1 | 17 | **0.0** | 0.0 | 0.0 | 0.1 | 0.0 | **2.6** | 2.7 | 3.2 | 2.9 | 4.9 |
| | 10 | 22 | **48.5** | 55.2 | 66.5 | 1712.7 | 3300.9 | **33.4** | 36.1 | 40.7 | 722.7 | 1742.0 |
| socfb-UGA50 | 1 | 53 | 0.2 | 0.2 | **0.2** | 12.1 | 0.5 | 4.1 | **4.0** | 5.0 | 7.7 | 7.5 |
| | 10 | 58 | **27.5** | 90.5 | 82.2 | OOT | 5319.5 | **10.4** | 23.6 | 31.4 | OOT | 4950.6 |
| | 20 | 61 | **21325.2** | OOT | OOT | OOT | OOT | **3601.1** | OOT | OOT | OOT | OOT |

### 5.2 *The Effect of the Upper Bound*

We empirically evaluate the upper bounds by comparing DnBk (which uses packing-and-coloring bound with conflict pairs) with the following algorithms.

- DnBk$_P$, the DnBk that uses the packing bound, see Bounding Rule 1.
- DnBk$_C$, the DnBk that uses the coloring bound, see Bounding Rule 2.
- DnBk$_S$, the DnBk that uses the *Sorting* Bound in [8].
- DnBk$_{Club}$, the DnBk that uses the *Club* bound in [18].

Due to the space limit, we show the computation time and the nodes of each algorithm for 12 benchmark graphs in Tab. 1. Here, the number of nodes refers to the number of branches in the search tree produced by the algorithm. We can see that for all instances, DnBk has fewer nodes than DnBk$_S$, which in turn has fewer nodes than DnBk$_P$ and DnBk$_C$. This matches the dominance relations that we conclude in Fig. 4. For instances like rt-retweet-crawl and tech-as-skitter with $k$=1, all algorithms can solve them within 10 seconds, the other variant could be faster than DnBk. This shows that the time overhead of computing the bound only affects easy instances. The number of nodes is fewer than that of DnBk$_{Club}$ in most cases, implying that our new bound is empirically better than Club. Also, DnBk is faster than this algorithm.

## 6 Conclusion

In the paper, we discussed both theoretical and practical aspects of solving the maximum $k$-defective clique problem. We studied a branching algorithm that has a better exponential time complexity than the existing ones when $k$ is constant. We also investigated a new bound that hybrids existing two bounds with the idea of conflict vertices. The tightness of the bounds is analyzed based on the dominance relationships. We note that the conflict relationship of vertices, which was first formalized in this paper, exists in many graph search problems, such as the maximum clique problem, and thus can be potentially extended in the future.

## Acknowledgements

This work was supported by Natural Science Foundation of Sichuan Province of China under grants 2023NSFSC1415 and 2023NSFSC0059, and National Natural Science Foundation of China under grants 61972070 and 62372095. The second author is also funded by China Postdoctoral Science Foundation under grant 2022M722815.

## References

[1] M. Azaouzi, D. Rhouma, and L. Ben Romdhane. Community detection in large-scale social networks: state-of-the-art and future directions. *Social Network Analysis and Mining*, 9(1):23, 2019.

[2] P. Bedi and C. Sharma. Community detection in social networks. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 6(3): 115–135, 2016.

[3] R. Behar and S. Cohen. Finding all maximal connected s-cliques in social networks. In *EDBT*, pages 61–72, 2018.

[4] A. Bettinelli, V. Cacchiani, and E. Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29(3):457–473, 2017.

[5] V. Boginski, S. Butenko, and P. M. Pardalos. Statistical analysis of financial networks. *Computational statistics & data analysis*, 48(2): 431–443, 2005.

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

[7] L. Chang. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 529–538, 2019.

[8] L. Chang. Efficient maximum k-defective clique computation with improved time complexity. *Proceedings of the ACM on Management of Data*, 1(3):1–26, 2023.

[9] L. Chang. Maximum defective clique computation: Improved time complexities and practical performance. *arXiv preprint arXiv:2403.07561*, 2024.

[10] X. Chen, Y. Zhou, J.-K. Hao, and M. Xiao. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research*, 127:105131, 2021.

[11] A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1272–1281. ACM, 2018.

[12] Q. Dai, R.-H. Li, M. Liao, and G. Wang. Maximal defective clique enumeration. *Proceedings of the ACM on Management of Data*, 1(1): 1–26, 2023.

[13] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 16–25, 2007.

[14] J. Gao, Z. Xu, R. Li, and M. Yin. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10174–10183, 2022.

[15] T. Gschwind, S. Irnich, and I. Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 234:131–138, 2018.

[16] E. Harley, A. Bonner, and N. Goodman. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics*, 17(6):487–494, 2001.

[17] J. Håstad. Clique is hard to approximate withinn 1- $\varepsilon$. *Acta Mathematica*, 182(1):105–142, 1999.

[18] M. Jin, J. Zheng, and K. He. Kd-club: An efficient exact algorithm with new coloring-based upper bound for the maximum k-defective clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20735–20742, 2024.

[19] S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

[20] C.-M. Li, H. Jiang, and F. Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017.

[21] C. Luo, Y. Zhou, Z. Wang, and M. Xiao. A faster branching algorithm for the maximum $k$-defective clique problem, 2024. URL https://arxiv.org/abs/2407.16588.

[22] S. Matsugu, Y. Fujiwara, and H. Shiokawa. Uncovering the largest community in social networks at scale. In *IJCAI*, pages 2251–2260, 2023.

[23] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

[24] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. URL http://networkrepository.com.

[25] S. Trukhanov, C. Balasubramaniam, B. Balasundaram, and S. Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, 2013.

[26] Z. Wang, Y. Zhou, M. Xiao, and B. Khoussainov. Listing maximal k-plexes in large real-world graphs. In *Proceedings of the ACM Web Conference 2022*, pages 1517–1527, 2022.

[27] Z. Wang, Y. Zhou, C. Luo, and M. Xiao. A fast maximum k-plex algorithm parameterized by the degeneracy gap. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5648–5656. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/627. URL https://doi.org/10.24963/ijcai.2023/627. Main Track.

[28] N. Wünsche. Mind the gap when searching for relaxed cliques. Master's thesis, Technische Universität Berlin, 2021.

[29] M. Xiao and H. Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.

[30] M. Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264. ACM, 1978.

[31] H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.

[32] Y. Zhou, S. Hu, M. Xiao, and Z.-H. Fu. Improving maximum k-plex solver via second-order reduction and graph color bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12453–12460, 2021.