



# Revisiting the Maximum Defective Clique Problem: Faster Branching and a Tighter Upper Bound

Kewu Yang

Harbin Institute of Technology, Shenzhen  
220110422@stu.hit.edu.cn

Shengxin Liu\*

Harbin Institute of Technology, Shenzhen  
sxliu@hit.edu.cn

Kaiqiang Yu\*

State Key Laboratory of Novel Software Technology,  
Nanjing University  
kaiqiang.yu@nju.edu.cn

Zhaoquan Gu\*

Harbin Institute of Technology, Shenzhen  
PengCheng Laboratory  
guzhaoquan@hit.edu.cn

## ABSTRACT

The  $k$ -defective clique model relaxes the strict completeness constraint of the traditional clique by allowing up to  $k$  missing edges, providing a robust formulation for detecting cohesive structures in noisy graphs. Consequently, the maximum  $k$ -defective clique problem has attracted significant attention. State-of-the-art exact algorithms predominantly adopt the branch-and-bound framework, which recursively partitions the current problem instance (or branch) into two sub-problems via a branching procedure, until each sub-problem becomes trivially solvable. However, this strategy often leads to excessive branching by overlooking intermediate sub-problems that are non-trivial yet efficiently solvable. While recent studies have attempted to refine branching procedures, they fail to address this structural redundancy. To address this, we propose BBRes, a framework that incorporates a novel early termination strategy into the recursive branching process. By employing a specialized polynomial-time solver to identify and resolve tractable sub-instances, BBRes effectively avoids redundant branching steps. Additionally, we design a tailored branching strategy that synergizes with this termination mechanism. As a result, BBRes achieves an improved theoretical worst-case time complexity. To enhance practical performance, we propose a tighter upper bound based on a novel double graph coloring method integrated with max-flow techniques, which is orthogonal to the branching framework. Extensive experiments show that BBRes achieves at least 2X speedup over state-of-the-art methods on a substantial fraction of the datasets.

## PVLDB Reference Format:

Kewu Yang, Kaiqiang Yu, Shengxin Liu, and Zhaoquan Gu. Revisiting the Maximum Defective Clique Problem: Faster Branching and a Tighter Upper Bound. PVLDB, 19(9): 2439 – 2451, 2026.  
doi:10.14778/3819518.3819562

## PVLDB Artifact Availability:

\*Corresponding authors.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org). Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.  
Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.  
doi:10.14778/3819518.3819562

The source code, data, and/or other artifacts have been made available at <https://github.com/Thaumaturge2020/BBRes/>.

## 1 INTRODUCTION

Graphs are widely used to model relationships between entities in various applications, such as social media, biological science, and e-commerce. Extracting cohesive/dense subgraphs from large real-world graphs is a fundamental problem in graph analytics. Such cohesive subgraphs often carry interesting information, which can facilitate various tasks across different domains. Examples include finding communities in social networks [5, 21], detecting anomalies in financial networks or social media [1, 6, 32, 52], and discovering biologically relevant functional groups in biological networks [44].

One notable cohesive subgraph model is the clique, which requires that every pair of vertices is connected by an edge [9, 15, 20, 35, 36, 39, 41, 42]. However, the strict completeness requirement of cliques is often too restrictive for real-world applications due to data quality issues caused by noise and missing values. To solve this issue, recent studies have relaxed the clique model and proposed various clique relaxation models, including quasi-clique [30, 38, 47, 53, 54],  $k$ -plex [13, 24, 40, 46, 48, 55],  $k$ -club [7], and  $k$ -defective clique [11, 12, 14, 17, 23, 29, 34, 51]. In this paper, we focus on the  $k$ -defective clique model, which relaxes the clique model by allowing up to  $k$  missing edges, where  $k$  is a positive integer. Conceptually, a  $k$ -defective clique is a subgraph induced by a set of vertices  $S$  that contains at least  $\binom{|S|}{2} - k$  edges. We study the problem of finding the maximum  $k$ -defective clique, i.e., the one with the largest number of vertices, which has been used for various applications such as interaction prediction in biological networks [51], cluster detection [18], and social network analysis [26].

**State-of-the-art methods.** Recently, a number of exact algorithms have been proposed in the literature for finding the maximum  $k$ -defective clique [11, 12, 14, 17, 23, 28, 29, 34]. The majority of them follow a similar *branch-and-bound* (BB) framework [11, 12, 17]. Specifically, this framework recursively partitions the problem into two sub-problems on *smaller* subgraphs via a *branching* method, continuing until each sub-problem can be solved trivially (i.e., the corresponding subgraph is already a  $k$ -defective clique). Each sub-problem corresponds to a branch. To improve efficiency, reduction rules are developed to prune unpromising branches that cannot contain the largest  $k$ -defective clique. Among these reductions,

upper-bound-based pruning is often the most powerful in practice. The rationale is that the algorithm computes an upper bound on the size of any  $k$ -defective clique derivable from the current branch, and it safely prunes the branch if this upper bound is not larger than the size of the largest  $k$ -defective clique found so far. Therefore, tight upper bounds are preferred to enhance practical performance. In terms of theoretical complexity, we note that recent works focus on sharpening the branching method to improve the worst-case time complexity. The best known complexity is  $O^*(\gamma_k^n)$  [17], where  $O^*$  suppresses polynomial factors,  $n$  is the number of vertices, and  $\gamma_k$  is the largest real root of the equation  $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ .

While some recent approaches, such as DnBK [34] and WODC [28], attempt to depart from the standard BB framework to reduce the exponential base of the time complexity, these theoretical improvements come at a prohibitive cost. Specifically, they introduce a substantial overhead factor scaled by  $(\delta\Delta)^{\Theta(k)}$ , where  $\delta$  and  $\Delta$  denote the degeneracy and maximum degree of the graph, respectively. In real-world graphs where  $\Delta$  is typically large, this overhead explodes as  $k$  increases, rendering such methods computationally impractical for difficult instances. Consequently, existing state-of-the-art algorithms, whether adhering to the standard BB framework or alternative paradigms, still suffer from significant efficiency bottlenecks in both theory and practice.

**Our method.** In this paper, instead of solely refining the branching strategies as existing methods do, we focus on *both the termination strategy and the branching strategy*. Specifically, we observe that existing methods terminate the recursive branching procedure of generating new branches only when the current sub-problem can be solved *trivially*, which incurs an excessive number of recursive calls to reach such a trivial branch. As a result, they generate an excessive number of branches. Motivated by this, we propose to terminate the recursive branching procedure once the current sub-problem can be solved efficiently (e.g., in polynomial time) via a non-recursive solver, thereby generating fewer branches. We call this strategy the *early termination strategy*. To implement this strategy, we first formulate *non-trivial* input-restricted problems (of finding the largest  $k$ -defective clique) called MissingTwoDeg, which corresponds to a collection of non-trivial branches. Then, we develop a greedy method called IRSolver to solve MissingTwoDeg instances in polynomial time. Based on IRSolver, our early termination strategy halts the recursive branching and invokes IRSolver when the current branch corresponds to an input-restricted problem. Furthermore, to fully exploit the potential of our early termination strategy, we design a new branching strategy called **BS-three**. Compared to existing strategies, **BS-three** better complements the early termination strategy as it tends to generate fewer branches. With the early termination strategy and new branching strategy, our branch-and-bound algorithm called BBRes achieves the worst-case time complexity of  $O^*(\lambda_k^n)$ , where  $\lambda_k$  is the largest real root of the equation  $x^{k+4} - 2x^{k+3} + x^3 - x + 1 = 0$ . For example,  $\lambda_k = 1.381, 1.705, 1.867$  when  $k = 1, 2, 3$ , respectively. We remark that, compared to the state-of-the-art method MDC [17], our BBRes further improves the worst-case time complexity from  $O^*(\gamma_k^n)$  to  $O^*(\lambda_k^n)$ , where  $\lambda_k$  is *strictly smaller than*  $\gamma_k$  for  $k \geq 1$ .

In addition, to further boost practical performance, we propose a new upper bound on the size of the  $k$ -defective clique to be found

in a branch, by leveraging double graph coloring and max-flow techniques. Specifically, we perform graph coloring *twice*, assigning each vertex a pair of colors. Then, based on the double-coloring information, we construct a constrained max-flow problem to compute the upper bound. Compared to existing color-based upper bounds [12, 14, 17] that also utilize graph coloring, our upper bound **UB-Double** is *practically tighter*, as it leverages more topological information by applying graph coloring twice and incorporating the max-flow technique. We remark that this upper bound is orthogonal to the branch-and-bound framework and can also be applied to existing methods [11, 12, 14, 17, 23, 29].

**Contributions.** We summarize our main contributions as follows.

- We propose a new branch-and-bound algorithm called BBRes which is based on our newly developed early termination strategy and branching strategy. BBRes has the worst-case time complexity of  $O^*(\lambda_k^n)$ , improving upon the state-of-the-art method [17] since  $\lambda_k$  is strictly smaller than  $\gamma_k$  for  $k \geq 1$ . (Section 4)
- We further propose a new upper bound of the size of  $k$ -defective cliques to be found in a branch based on the double graph coloring technique and the max-flow technique. Our upper bound is tighter than the existing ones in practice. (Section 5)
- Finally, we conduct extensive experiments on a collection of 139 real-world graphs. The results show that our BBRes outperforms other baselines, including kDC2 [12], MDC [17], DnBK [34], and WODC [28], by solving more problem instances. (Section 6)

For the rest of the paper, we define the problem in Section 2, present state-of-the-art methods in Section 3, review related work in Section 7, and conclude the paper in Section 8.

## 2 PRELIMINARIES

We consider an unweighted and undirected graph  $G = (V, E)$ , where  $V$  is the vertex set and  $E$  is the edge set. Let  $n = |V|$  and  $m = |E|$  denote the number of vertices and edges, respectively. We define  $\bar{E}$  as the set of edges that are missing in  $G$  (referred to as **non-edges**), i.e.,  $\bar{E} = \{(u, v) \in V \times V \mid u \neq v \text{ and } (u, v) \notin E\}$ .

Given a vertex  $v$  in  $V$ , let  $N_G(v)$  (resp.  $\bar{N}_G(v)$ ) be the set of neighbors (resp. non-neighbors) of  $v$  in  $G$ ; formally,  $N_G(v) = \{u \in V \mid (u, v) \in E\}$  (resp.  $\bar{N}_G(v) = \{u \in V \mid (u, v) \in \bar{E}\}$ ). Accordingly, we define the degree of  $v$  in  $G$  as  $d_G(v) = |N_G(v)|$ , and its non-degree as  $\bar{d}_G(v) = |\bar{N}_G(v)|$ . We note that a vertex is neither a neighbor nor a non-neighbor of itself. Given a vertex subset  $S \subseteq V$ , we use  $G[S]$  to denote the subgraph of  $G$  induced by  $S$ , i.e.,  $G[S]$  consists of the set of vertices  $S$  and the set of edges  $\{(u, v) \in E \mid u, v \in S\}$ . All subgraphs considered in this paper are induced subgraphs. Given a subgraph  $g$  of  $G$ , we use  $V(g)$ ,  $E(g)$ , and  $\bar{E}(g)$  to denote its sets of vertices, edges, and non-edges, respectively.

**Abbreviations.** For simplicity, we omit the subscript  $G$  from the notation when the context is clear. Furthermore, for  $S \subseteq V$ , we abbreviate  $N_{G[S]}(\cdot)$  as  $N_S(\cdot)$ ,  $d_{G[S]}(\cdot)$  as  $d_S(\cdot)$ , and  $E(G[S])$  as  $E(S)$ . Similarly, we abbreviate  $\bar{N}_{G[S]}(\cdot)$ ,  $\bar{d}_{G[S]}(\cdot)$ , and  $\bar{E}(G[S])$  as  $\bar{N}_S(\cdot)$ ,  $\bar{d}_S(\cdot)$ , and  $\bar{E}(S)$ , respectively.

In this paper, we focus on the concept of  $k$ -defective clique.

**Definition 2.1 ( $k$ -Defective Clique [51]).** A graph  $g$  is said to be a  $k$ -defective clique if it contains at most  $k$  non-edges, i.e.,  $|\bar{E}(g)| \leq k$  or equivalently,  $|E(g)| \geq \frac{|V(g)|(|V(g)|-1)}{2} - k$ .

Clearly, a 0-defective clique is simply a clique, in which every pair of vertices is adjacent. In addition, we note that the  $k$ -defective clique satisfies the *hereditary* property, i.e., any subgraph of a  $k$ -defective clique is also a  $k$ -defective clique [37]. We now formally state the problem studied in this paper.

**PROBLEM DEFINITION (MAXIMUM  $k$ -DEFECTIVE CLIQUE [11, 12, 14, 17, 23, 34]).** Given a graph  $G = (V, E)$  and a positive integer  $k$ , the maximum  $k$ -defective clique problem aims to find the maximum  $k$ -defective clique in  $G$ , i.e., the  $k$ -defective clique with the largest number of vertices.

We note that the maximum  $k$ -defective clique in  $G$  may not be unique, and the maximum  $k$ -defective clique problem is known to be NP-hard [33, 43]. Moreover, although a larger  $k$  relaxes the feasibility constraint, it also enlarges the family of feasible solutions. In particular, the number of maximal  $k$ -defective cliques can grow exponentially with  $k$  [11, 12, 17], resulting in a larger search space and making the problem harder in practice.

### 3 STATE-OF-THE-ART ALGORITHMS

The state-of-the-art algorithms for the maximum  $k$ -defective clique predominantly adopt the *branch-and-bound* (BB) framework [11, 12, 17]. The core idea is to recursively partition the current problem instance (i.e., search space), which aims to find the largest  $k$ -defective clique, into two smaller sub-problems via *branching* until each of them can be solved trivially. Specifically, a problem instance (or branch) is represented as a triple  $(g, S, C)$ , where:

- Graph  $g$  is a subgraph of the input graph  $G$  induced by the vertex set  $S \cup C$ , i.e.,  $g = G[S \cup C]$ ;
- Partial set  $S$  is a set of vertices that induces a  $k$ -defective clique and must be contained in the largest  $k$ -defective clique found within this branch;
- Candidate set  $C$  is a set of vertices that will be considered to be included in  $S$ .

Solving a branch  $(g, S, C)$  means finding in this branch the largest  $k$ -defective clique  $g^*$  that (1) includes all vertices in  $S$  and (2) is a subgraph of  $g$ , i.e.,  $S \subseteq V(g^*)$  and  $V(g^*) \subseteq C \cup S$ . Clearly, solving the branch  $(G, \emptyset, V)$  finds the maximum  $k$ -defective clique in  $G$ .

To solve a branch  $(g, S, C)$ , the framework recursively partitions the branch into two sub-branches via branching. In particular, it selects a vertex  $v^*$  called *pivot* from the candidate set  $C$ , and generates two sub-branches. The first one  $(g_1, S \cup \{v_p\}, C \setminus \{v_p\})$  removes the pivot  $v_p$  from  $C$  to  $S$  (which aims to explore the largest  $k$ -defective clique that includes  $v_p$ ); the second one  $(g_2, S, C \setminus \{v_p\})$  discards the pivot  $v_p$  from  $C$  (which aims to explore the largest  $k$ -defective clique that excludes  $v_p$ ). Here,  $g_1$  and  $g_2$  are the subgraphs induced by the updated partial and candidate sets in each sub-branch. By recursively solving both sub-branches, the framework ensures that all possibilities are explored and thus solves the original branch.

**Summary.** We summarize the BB framework in Algorithm 1. Specifically, the algorithm maintains the largest  $k$ -defective clique  $g^*$  seen so far during the recursion (Line 1). To boost the practical performance, it utilizes *the diameter-two property* ([14]) of large  $k$ -defective cliques (i.e., any  $k$ -defective clique with at least  $k + 2$  vertices has diameter at most 2) and involves the following stages.

- **Stage-I** adopts the diameter-two property for pruning by assuming that the largest  $k$ -defective clique is of size at least

---

**Algorithm 1:** The state-of-the-art branch-and-bound (BB) framework [11, 12, 17]

---

**Input:** A graph  $G = (V, E)$  and a positive integer  $k$   
**Output:** The maximum  $k$ -defective clique  $g^*$   
 /\* Stage-I: With the diameter-two property \*/  
 1 Let  $g^* \leftarrow \emptyset$  be the largest  $k$ -defective clique seen so far;  
 2 Let  $V = \{v_1, v_2, \dots, v_n\}$  be a degeneracy ordering of vertices in  $G$ ;  
 3 **foreach**  $v_i \in \{v_1, v_2, \dots, v_n\}$  **do**  
 4      $G_{v_i} \leftarrow$  the subgraph of  $G$  induced by  
        $N^{\leq 2}(v_i) \cap \{v_i, v_{i+1}, \dots, v_n\}$ ;  
 5      $\text{BB\_Rec}(G_{v_i}, \{v_i\}, V(G_{v_i}) \setminus \{v_i\})$ ;  
 /\* Stage-II: Without the diameter-two property \*/  
 6 **if**  $|V(g^*)| < k + 1$  **then**  $\text{BB\_Rec}(G, \emptyset, V)$ ;  
 7 **return**  $g^*$ ;  
 8 **Procedure**  $\text{BB\_Rec}(g, S, C)$   
    /\* Reduction \*/  
 9      $UB \leftarrow$  an upper bound of the branch;  
 10     **if**  $UB \leq |V(g^*)|$  **then return**;  
 11     Refine  $C$  (and  $g$ ) by applying reduction rules;  
    /\* Termination \*/  
 12     **if**  $g$  is a  $k$ -defective clique **then**  
 13         **if**  $|V(g)| > |V(g^*)|$  **then**  $g^* \leftarrow g$ ;  
 14         **return**;  
    /\* Branching \*/  
 15      $v_p \leftarrow$  a pivot selected from  $C$ ;  
 16     Construct  $g_1$  and  $g_2$  based on  $v_p$ ,  $S$ , and  $C$ ;  
 17      $\text{BB\_Rec}(g_1, S \cup \{v_p\}, C \setminus \{v_p\})$ ;  
 18      $\text{BB\_Rec}(g_2, S, C \setminus \{v_p\})$ ;

---

$k + 2$  (Lines 2-5). Specifically, it first computes a degeneracy ordering of the vertices in  $G$  (Line 2), which can be done efficiently in  $O(m)$  by the peeling algorithm [4]. Then, it divides the problem of finding the maximum  $k$ -defective clique into  $n$  sub-problems (Lines 3-5). The  $i$ -th one aims to find the maximum  $k$ -defective clique that (1) includes  $v_i$  and (2) is a subgraph of  $G[\{v_i, v_{i+1}, \dots, v_n\}]$  by invoking the BB procedure with the branch  $(G_{v_i}, \{v_i\}, V(G_{v_i}) \setminus \{v_i\})$  (Lines 3-5). In particular,  $G_{v_i}$  is the subgraph of  $G$  induced by  $v_i$ 's two-hop neighbors in  $\{v_i, v_{i+1}, \dots, v_n\}$ , i.e.,  $N^{\leq 2}(v_i) \cap \{v_i, v_{i+1}, \dots, v_n\}$  (note that a  $k$ -defective clique with at least  $k + 2$  vertices has diameter of at most 2 and thus the largest one containing  $v_i$  is a subset of  $N^{\leq 2}(v_i)$ ). Clearly, if the found largest  $k$ -defective clique  $g^*$  is of at least  $k + 1$  vertices,  $g^*$  is guaranteed to be a maximum  $k$ -defective clique of  $G$ .

- **Stage-II** continues the search for the maximum  $k$ -defective clique when Stage-I does not succeed, i.e., when  $g^*$  obtained in Stage-I has fewer than  $k + 1$  vertices (Line 6). To this end, it invokes the BB procedure with the branch  $(G, \emptyset, V)$ .

In addition, the BB procedure called  $\text{BB\_Rec}$  is summarized in Lines 8-18. We note that the recursive procedure terminates when  $g$  becomes a  $k$ -defective clique since  $g$  is the largest  $k$ -defective clique in the branch (Lines 12-14). To reduce the search space, the framework employs two techniques as follows.

- **Upper-bound-based reductions** (Lines 9-10). It first computes an upper bound of the size of the largest  $k$ -defective clique in the

branch. Then, the branch can be terminated if the upper bound is no larger than the largest  $k$ -defective clique seen so far.

- **Pivot-based branching** (Lines 15-18). The number of branches generated by the pivot-based branching depends on the pivot selection strategy (Line 15). We note that recent studies have explored various strategies for pivot selection to reduce the number of branches [11, 12, 17].

For brevity, we do not review the detailed implementation of these techniques in state-of-the-art algorithms.

**Time complexity.** Following Algorithm 1, the state-of-the-art studies have focused on improving the worst-case time complexity by sharpening the pivot selection strategy [11, 12, 17]. As a result, the latest algorithm has the time complexity of  $O^*(\gamma_k^n)$  [17], where  $O^*$  ignores the polynomial factors and  $\gamma_i$  is the largest real root of the equation  $x^{i+3} - 2x^{i+2} + x^2 - x + 1 = 0$ .

**Remark.** It is worth noting that two recent algorithms, DnBK [34] and WODC [28], employ frameworks distinct from the above BB framework to successfully reduce the exponential base in time complexity. However, this improvement comes at the cost of introducing a substantial overhead factor scaled by  $(\delta\Delta)^{\Theta(k)}$ , where  $\delta$  is the degeneracy of the input graph  $G$  and  $\Delta$  is the maximum degree of  $G$ . This factor becomes prohibitively large when  $k$  is not a constant, severely limiting scalability. As demonstrated in our experiments (Section 6), both methods are less competitive compared to our proposed algorithm. We defer the detailed complexity analysis of both algorithms to the related work section (Section 7).

## 4 A NEW BRANCH-AND-BOUND FRAMEWORK: BBRES

### 4.1 BBRes: Motivation and Overview

**Motivation.** The existing BB framework recursively partitions the current problem instance  $(g, S, C)$  into two sub-problems via branching until each of them can be solved *trivially*, i.e.,  $g$  becomes a  $k$ -defective clique. As a result, the branching procedure, which is biased towards generating trivial instances, produces an exponentially large number of branches in the worst case (e.g.,  $O(\gamma_k^n)$  [17]), thereby dominating the time complexity. To reduce the number of branches, we propose to *guide the branching procedure towards non-trivial yet solvable problem instances*. The rationale is to terminate the recursive branching procedure early whenever the current problem instance can be solved efficiently (e.g., in polynomial time) by a dedicated solver, thus generating fewer branches.

**Overview.** Motivated by the above, we develop a new BB framework called BBRes, as summarized in Algorithm 2. BBRes differs from Algorithm 1 mainly in the *early termination strategy* (Lines 12-15) and the *branching strategy* (Lines 16-18). Specifically, we first formulate an input-restricted problem instance (i.e., a branch  $(g, S, C)$  satisfying certain constraints), which is *non-trivial* (i.e.,  $g$  can be a non- $k$ -defective clique). Note that such input-restricted instances will be solved recursively via branching in Algorithm 1. However, we observe that these instances can be solved efficiently in polynomial time by a non-recursive method called IRSolver in Section 4.2. Based on this observation, our BBRes can terminate the recursive branching procedure once it reaches an input-restricted instance, i.e., satisfying the termination conditions, and

---

### Algorithm 2: Our framework BBRes

---

**Input:** A graph  $G = (V, E)$  and a positive integer  $k$   
**Output:** The maximum  $k$ -defective clique  $g^*$   
 /\* Stage-I: With the diameter-two property \*/  
 1 Let  $g^* \leftarrow \emptyset$  be the largest  $k$ -defective clique seen so far;  
 2 Let  $V = \{v_1, v_2, \dots, v_n\}$  be a degeneracy ordering of vertices in  $G$ ;  
 3 **foreach**  $v_i \in \{v_1, v_2, \dots, v_n\}$  **do**  
 4      $G_{v_i} \leftarrow$  the subgraph of  $G$  induced by  
        $N^{\leq 2}(v_i) \cap \{v_i, v_{i+1}, \dots, v_n\}$ ;  
 5     BBRes\_Rec( $G_{v_i}, \{v_i\}, V(G_{v_i}) \setminus \{v_i\}$ );  
 /\* Stage-II: Without the diameter-two property \*/  
 6 **if**  $|V(g^*)| < k + 1$  **then** BBRes\_Rec( $G, \emptyset, V$ );  
 7 **return**  $g^*$ ;  
 8 **Procedure** BBRes\_Rec( $g, S, C$ )  
    /\* Reductions (Section 5) \*/  
     $UB \leftarrow$  an upper bound of the branch;  
    **if**  $UB \leq |V(g^*)|$  **then return**;  
    Refining  $C$  (and  $g$ ) by applying reduction rules;  
    /\* Early Termination Strategy (Section 4.2) \*/  
    **if**  $g$  satisfies the early termination conditions, i.e., **Condition 1**  
       **or Condition 2** in Section 4.2 **then**  
        $g_{opt} \leftarrow$  IRSolver( $g, S, C$ );  
       **if**  $|V(g_{opt})| > |V(g^*)|$  **then**  $g^* \leftarrow g_{opt}$ ;  
       **return**;  
    /\* Branching (Section 4.3) \*/  
     $v_p \leftarrow$  a pivot selected from  $C$  based on our new strategy;  
    BBRes\_Rec( $g_1, S \cup \{v_p\}, C \setminus \{v_p\}$ );  
    BBRes\_Rec( $g_2, S, C \setminus \{v_p\}$ );

---

uses IRSolver to solve the branch. Furthermore, we observe that the existing branching strategies are not suitable for our BBRes since they are designed based on the framework of recursively branching to trivial branches. To boost the performance of BBRes, we further propose a new branching strategy in Section 4.3. With this newly-designed branching strategy, BBRes achieves better worst-case time complexity. In addition, we remark that we propose a novel upper bound (Lines 9-11) based on graph coloring and max-flow techniques (details will be discussed in Section 5), which is orthogonal to the framework and further enhances practical performance. We remark that our BBRes also employs those existing techniques, including reductions and pre-processing techniques for reducing the input graph [12, 17, 28], that are orthogonal to the framework.

### 4.2 BBRes: Early Termination Strategy

Consider a problem instance  $(g, S, C)$ . Note that a problem instance  $(g, S, C)$  is said to be *trivial* if  $g$  is a  $k$ -defective clique. Our early termination strategy is motivated by the observation that certain non-trivial instances admit a particularly simple structure. Specifically, if every vertex in the candidate set  $C$  has at most two non-neighbors within  $C$ , then in the complement graph of  $G[C]$ , every vertex has degree at most 2. Hence, each connected component of the complement graph is a path, a cycle, or an isolated vertex. This structural property enables an efficient algorithm for the remaining subproblem. Thus, we define the input-restricted problem instances, namely MissingTwoDeg, as follows.

---

**Algorithm 3:** Our greedy method IRSolver

---

**Input:** A MissingTwoDeg problem instance  $(g, S, C)$  and a positive integer  $k$

**Output:** The maximum  $k$ -defective clique  $g_{opt}$  that is a subgraph of  $g$  and contains  $S$

```
1 if  $g$  is a  $k$ -defective clique then return  $g$  as  $g_{opt}$  directly;
2  $S_{opt} \leftarrow S, C_{temp} \leftarrow C$ ;
3 while true do
4    $\Gamma_{min} \leftarrow \{v \in C_{temp} \mid \forall w \in C_{temp}, \bar{d}_{S_{opt}}(v) \leq \bar{d}_{S_{opt}}(w)\}$ ;
5   if Case 1:  $\min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v) \leq 1$  then
6      $v^* \leftarrow \arg \min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v)$ ;
7   else if Case 2:  $\min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v) = 2$  then
8      $v^* \leftarrow \arg \min_{v \in \Gamma_{min}} \bar{d}_{\Gamma_{min}}(v)$ ;
9   if  $G[S_{opt} \cup \{v^*\}]$  is not a  $k$ -defective clique then break;
10   $C_{temp} \leftarrow C_{temp} \setminus \{v^*\}; S_{opt} \leftarrow S_{opt} \cup \{v^*\}$ ;
11 return  $G[S_{opt}]$  as the solution  $g_{opt}$ ;
```

---

*Definition 4.1 (MissingTwoDeg problem instance).* A problem instance  $(g, S, C)$  is said to be a MissingTwoDeg problem if it satisfies one of the following conditions:

- **Condition 1.**  $g$  is a  $k$ -defective clique, i.e.,  $|\bar{E}(g)| \leq k$ ;
- **Condition 2.**  $g$  is not a  $k$ -defective clique, and for any vertex  $v$  in  $C$ ,  $v$  has at most 2 non-neighbors in  $C$ , i.e.,  $|\bar{E}(g)| > k$  and  $\forall v \in C, \bar{d}_C(v) \leq 2$ .

Condition 1 corresponds to the trivial termination case, whereas instances satisfying Condition 2 are non-trivial but remain tractable due to the special structure of the complement graph induced by  $C$ . Thus, terminating the branching procedure once a MissingTwoDeg instance is reached can reduce the number of generated branches while still allowing the remaining subproblem to be solved efficiently. One remaining question is how to solve a MissingTwoDeg problem efficiently in polynomial time. To this end, we introduce a greedy method called IRSolver.

**Overview of IRSolver.** If  $g$  is a  $k$ -defective clique, we can solve the problem by returning  $g$  directly. Otherwise, IRSolver solves the problem in a greedy manner, which runs in multiple rounds, as summarized in Algorithm 3. Specifically, it maintains two sets  $S_{opt}$  (initially as  $S$ ) and  $C_{temp}$  (initially as  $C$ ). The key idea is to iteratively select a vertex  $v^*$  from  $C_{temp}$  and move it from  $C_{temp}$  to  $S_{opt}$  while maintaining that  $G[S_{opt}]$  is a  $k$ -defective clique until this is no longer possible (Lines 3-10, details will be discussed later). Finally, it returns  $G[S_{opt}]$  as the solution (Line 11). We then elaborate on the details regarding the greedy strategy of selecting  $v^*$ , the correctness, and the time complexity analysis.

**Greedy strategy of IRSolver.** At each round (Lines 3-10), IRSolver selects a vertex  $v^*$  from  $C_{temp}$  based on the following greedy strategy. Specifically, let  $\Gamma_{min}$  be the set of vertices each of which has the smallest number of non-neighbors in  $S_{opt}$ , formally,

$$\Gamma_{min} = \{v \in C_{temp} \mid \forall w \in C_{temp}, \bar{d}_{S_{opt}}(v) \leq \bar{d}_{S_{opt}}(w)\}. \quad (1)$$

Then, we select vertex  $v^*$  from  $\Gamma_{min}$ . In general, there are two cases.

- **Case 1:**  $\min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v) \leq 1$ , i.e., there exists a vertex  $v$  in  $\Gamma_{min}$  that has the number of non-neighbors in  $C_{temp}$  smaller than 2. In this case, we select from  $\Gamma_{min}$  the vertex with the

smallest number of non-neighbors in  $C_{temp}$  as  $v^*$ , formally,

$$\text{Greedy strategy 1: } v^* \leftarrow \arg \min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v). \quad (2)$$

- **Case 2:**  $\min_{v \in \Gamma_{min}} \bar{d}_{C_{temp}}(v) = 2$ , i.e., each vertex in  $\Gamma_{min}$  has two non-neighbors in  $C_{temp}$ . Then, we select from  $\Gamma_{min}$  the vertex with the smallest number of non-neighbors in  $\Gamma_{min}$ , formally,

$$\text{Greedy strategy 2: } v^* \leftarrow \arg \min_{v \in \Gamma_{min}} \bar{d}_{\Gamma_{min}}(v). \quad (3)$$

We note that every vertex in  $C_{temp}$  has at most two non-neighbors in  $C_{temp}$  based on the definition of the MissingTwoDeg problem. Thus, the above strategy covers all possible cases. In addition, we remark that if there are multiple choices when selecting  $v^*$  in Equation (2) or Equation (3), e.g., more than one vertex in  $\Gamma_{min}$  has the smallest number of non-neighbors in  $C_{temp}$  or  $\Gamma_{min}$ ,  $v^*$  can be chosen arbitrarily from among them.

We also provide an example of IRSolver in Appendix B of our technical report [50].

**Correctness of IRSolver.** The correctness of IRSolver can be guaranteed by the following lemma.

**LEMMA 4.2.** *Given a MissingTwoDeg problem instance  $(g, S, C)$ , IRSolver finds the largest  $k$ -defective clique  $g_{opt}$  that is a subgraph of  $g$  and contains all vertices in  $S$ .*

**PROOF.** Note that if  $g$  is a  $k$ -defective clique, the problem can be directly solved by returning  $g$  (Line 1). In the following, we assume that  $g$  is not a  $k$ -defective clique. Clearly, IRSolver can terminate at Line 9 since  $S \cup C$  is a superset of  $S_{opt}$  and is not a  $k$ -defective clique. Thus, assume that IRSolver runs in  $\tau + 1$  rounds ( $\tau \geq 1$ ). At the  $i$ -th ( $1 \leq i \leq \tau$ ) round, it adds  $v_i^*$  to  $S_{opt}$ . Finally, it terminates at the  $(\tau + 1)$ -st round since  $S \cup \{v_1^*, \dots, v_{\tau+1}^*\}$  is no longer a  $k$ -defective clique. In addition, for  $0 \leq i \leq \tau + 1$ , we let

$$S_{opt}^i = S \cup \{v_1^*, \dots, v_i^*\} \text{ and } C_{temp}^i = C \setminus \{v_1^*, \dots, v_i^*\}. \quad (4)$$

$$\Gamma_{min}^i = \{v \in C_{temp}^i \mid \forall w \in C_{temp}^i, \bar{d}_{S_{opt}^i}(v) \leq \bar{d}_{S_{opt}^i}(w)\} \quad (5)$$

Note that  $S_{opt}^0$  and  $C_{temp}^0$  are  $S$  and  $C$ , respectively.

We first show that, when IRSolver terminates,  $g_{opt} = G[S_{opt}^\tau]$  is a maximal  $k$ -defective clique in  $g$ , i.e., any vertex in  $C \setminus V(g_{opt})$  cannot be added to  $g_{opt}$  to form a larger  $k$ -defective clique. The reasons are as follows. Consider the last round of IRSolver where  $G[S_{opt}^\tau \cup \{v_{\tau+1}^*\}]$  is no longer a  $k$ -defective clique. We note that  $v_{\tau+1}^*$  has the smallest number of non-neighbors in  $S_{opt}^\tau$  among other vertices in  $C_{temp}^\tau$  (since it is selected from  $\Gamma_{min}$ ). Therefore, for any vertex  $v$  in  $C_{temp}^\tau$ ,  $G[S_{opt}^\tau \cup \{v\}]$  is not a  $k$ -defective clique since  $|\bar{E}(S_{opt}^\tau \cup \{v\})| = |\bar{E}(S_{opt}^\tau)| + \bar{d}_{S_{opt}^\tau}(v) \geq |\bar{E}(S_{opt}^\tau)| + \bar{d}_{S_{opt}^\tau}(v_{\tau+1}^*) > k$ .

We then show that  $g_{opt}$  is the largest  $k$ -defective clique in  $g$  that contains  $S$  by contradiction. Note that  $g_{opt}$  contains  $S$  clearly since  $S_{opt}$  is a superset of  $S$ . Let  $g_{sol}$  be the largest  $k$ -defective clique in  $g$  that contains  $S$ . Assume that  $g_{opt}$  is not the largest  $k$ -defective clique that contains  $S$ , i.e.,  $|V(g_{sol})| > |V(g_{opt})|$ . Then, we will show that, by the following construction process starting with  $g_{sol}$ , there exists a largest  $k$ -defective clique  $g'$  containing  $S$  such that  $g_{opt}$  is a subgraph of  $g'$ , i.e.,  $V(g_{opt}) \subset V(g')$ , which contradicts to the fact that  $g_{opt}$  is a maximal  $k$ -defective clique in  $g$ .

Finally, to complete the proof, we introduce an iterative process to construct  $g'$  based on  $g_{opt}$  and  $g_{sol}$ . Let  $\langle v_1^*, v_2^*, \dots, v_\tau^* \rangle$  be the ordering of vertices in  $V(g_{opt}) \setminus S$ . The procedure has four steps.

- **Step 1: Initialization.** Initialize  $g'_0$  to be  $g_{sol}$  and  $i$  to be 0;
- **Step 2: Termination checking.** If  $g'_i$  contains all vertices in  $g_{opt}$ , i.e.,  $V(g'_i) \supseteq V(g_{opt})$ , set  $g' = g'_i$  and stop the process.
- **Step 3: Construction.** Find the vertex  $v_{o_i}^*$  with the smallest index  $o_i$  ( $1 \leq o_i \leq \tau$ ) that is in  $V(g_{opt}) \setminus S$  but not in  $g'_i$ . We can easily deduce that  $g'_i$  contains all vertices in  $S_{opt}^{o_i-1}$  and  $V(g'_i) \setminus S_{opt}^{o_i-1}$  is a subset of  $C_{temp}^{o_i-1}$ , formally,

$$S_{opt}^{o_i-1} \subseteq V(g'_i) \text{ and } V(g'_i) \setminus S_{opt}^{o_i-1} \subseteq C_{temp}^{o_i-1}. \quad (6)$$

Based on the above, we have

$$V(g'_i) = S_{opt}^{o_i-1} \cup \Psi_i \text{ where } \Psi_i = V(g'_i) \setminus S_{opt}^{o_i-1}. \quad (7)$$

We then have three cases to construct  $g'_{i+1}$  (note that  $\bar{d}_{\Psi_i}(v_{o_i}^*) \leq \bar{d}_C(v_{o_i}^*) \leq 2$  based on Definition 4.1).

- **Case 1:**  $\bar{d}_{\Psi_i}(v_{o_i}^*) \leq 1$ . Let  $v$  be an arbitrary vertex in  $\Psi_i$  if  $\bar{d}_{\Psi_i}(v_{o_i}^*) = 0$ ; otherwise if  $\bar{d}_{\Psi_i}(v_{o_i}^*) = 1$ , let  $v$  be  $v_{o_i}^*$ 's non-neighbor in  $\Psi_i$ . We construct  $g'_{i+1}$  by swapping vertex  $v$  with  $v_{o_i}^*$ , formally,  $g'_{i+1} = g[V(g'_i) \setminus \{v\} \cup \{v_{o_i}^*\}]$ . Clearly,  $g'_{i+1}$  is a  $k$ -defective clique since  $|\bar{E}(g'_{i+1})| = |\bar{E}(g'_i)| - \bar{d}_{g'_i}(v) + \bar{d}_{V(g'_i) \setminus \{v\}}(v_{o_i}^*) \leq |\bar{E}(g'_i)| - \bar{d}_{S_{opt}^{o_i-1}}(v) + \bar{d}_{S_{opt}^{o_i-1}}(v_{o_i}^*) + \bar{d}_{\Psi_i \setminus \{v\}}(v_{o_i}^*) \leq k$  (note that  $\bar{d}_{\Psi_i \setminus \{v\}}(v_{o_i}^*)$  is clearly equal to 0 and  $v_{o_i}^*$  has the smallest number of non-neighbors in  $S_{opt}^{o_i-1}$  among other vertices in  $C_{temp}^{o_i-1}$  based on our greedy strategy).
- **Case 2:**  $\bar{d}_{\Psi_i}(v_{o_i}^*) = 2$  and  $\bar{d}_{\Gamma_{min}^{o_i-1}}(v_{o_i}^*) \leq 1$ . In this case, all  $v_{o_i}^*$ 's non-neighbors in  $C_{temp}^{o_i-1}$  are in  $\Psi_i$  (note that both  $\Psi_i$  and  $\Gamma_{min}^{o_i-1}$  are subsets of  $C_{temp}^{o_i-1}$ ). Let  $u$  be a  $v_{o_i}^*$ 's non-neighbor in  $\Psi_i$  but not in  $\Gamma_{min}^{o_i-1}$  (note that such a vertex  $u$  exists based on the condition of this case). We construct  $g'_{i+1}$  by swapping vertex  $u$  with  $v_{o_i}^*$ , formally,  $g'_{i+1} = g[V(g'_i) \setminus \{u\} \cup \{v_{o_i}^*\}]$ . We can deduce that  $g'_{i+1}$  is a  $k$ -defective clique since (1)  $\bar{d}_{S_{opt}^{o_i-1}}(u)$  is strictly larger than  $\bar{d}_{S_{opt}^{o_i-1}}(v_{o_i}^*)$  (note that vertices in  $\Gamma_{min}^{o_i-1}$  has the smallest number of non-neighbors in  $S_{opt}^{o_i-1}$  and  $v_{o_i}^*$  is selected from  $\Gamma_{min}^{o_i-1}$  based on our greedy strategy while  $u$  is not in  $\Gamma_{min}^{o_i-1}$ ) and (2)  $|\bar{E}(g'_{i+1})| = |\bar{E}(g'_i)| - \bar{d}_{g'_i}(u) + \bar{d}_{V(g'_i) \setminus \{u\}}(v_{o_i}^*) \leq |\bar{E}(g'_i)| - \bar{d}_{S_{opt}^{o_i-1}}(u) + \bar{d}_{S_{opt}^{o_i-1}}(v_{o_i}^*) + \bar{d}_{\Psi_i \setminus \{u\}}(v_{o_i}^*) = |\bar{E}(g'_i)| - \bar{d}_{S_{opt}^{o_i-1}}(u) + \bar{d}_{S_{opt}^{o_i-1}}(v_{o_i}^*) + 1 \leq k$ .
- **Case 3:**  $\bar{d}_{\Psi_i}(v_{o_i}^*) = 2$  and  $\bar{d}_{\Gamma_{min}^{o_i-1}}(v_{o_i}^*) = 2$ . In this case, any vertex  $v$  in  $\Gamma_{min}^{o_i-1}$  has two non-neighbors in  $\Gamma_{min}^{o_i-1}$  based on our greedy strategy 2 and is adjacent to all vertices in  $C \setminus \Gamma_{min}^{o_i-1}$  based on Definition 4.1. Clearly, there exists a circle in the complementary graph of  $g[\Gamma_{min}^{o_i-1}]$  that contains  $v_{o_i}^*$ . W.l.o.g., let  $\langle w_0, w_1, \dots, w_c = w_0 \rangle$  ( $c \geq 2$ ) be the circle where vertex  $w_j$  is not adjacent to  $w_{j+1}$  ( $0 \leq j \leq c-1$ ) in  $g$ , and let  $w_0 = v_{o_i}^*$ . Let  $\{w_{x_1}, w_{x_2}, \dots, w_{x_t}\}$  be the vertices in  $\{w_0, \dots, w_c\} \cap \Psi_i$ , as illustrated in Figure 1. Clearly,  $w_0 = w_c$  is not in

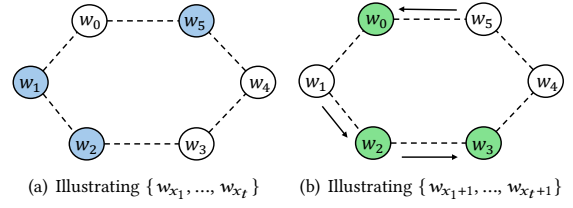


**Figure 1: An example to illustrate Case 3 in the proof of Lemma 4.2 with  $c = 6$  and  $\{w_{x_1}, \dots, w_{x_t}\} = \{w_1, w_2, w_5\}$  (indicated in blue); we have  $\{w_{x_1+1}, \dots, w_{x_t+1}\} = \{w_2, w_3, w_6 = w_0\}$  (indicated in green).**

$\{w_{x_1}, w_{x_2}, \dots, w_{x_t}\}$ . We construct  $g'_{i+1}$  by swapping the set of vertices  $\{w_{x_1}, w_{x_2}, \dots, w_{x_t}\}$  with  $\{w_{x_1+1}, w_{x_2+1}, \dots, w_{x_t+1}\}$ , i.e.,  $g'_{i+1} = g[V(g'_i) \setminus \{w_{x_1}, w_{x_2}, \dots, w_{x_t}\} \cup \{w_{x_1+1}, w_{x_2+1}, \dots, w_{x_t+1}\}]$ . We can deduce that  $w_0$  is in  $\{w_{x_1+1}, \dots, w_{x_t+1}\}$  since  $w_0$ 's non-neighbors  $w_1$  and  $w_{c-1}$  are both in  $\Psi_i$  in this case. Therefore,  $g'_{i+1}$  contains  $v_{o_i}^*$  and is a  $k$ -defective clique, since (1) every vertex in  $\{w_0, w_1, \dots, w_c\}$  belongs to  $\Gamma_{min}^{o_i-1}$ , and hence all these vertices have the same number of non-neighbors in  $S_{opt}^{o_i-1}$ ; and (2) we know that  $g_1 = g[\{w_{x_1}, w_{x_2}, \dots, w_{x_t}\}]$  and  $g_2 = g[\{w_{x_1+1}, w_{x_2+1}, \dots, w_{x_t+1}\}]$  are isomorphic under the mapping  $f : V(g_1) \rightarrow V(g_2)$  defined by  $f(w_j) = w_{j+1}$ . Indeed, for any  $(w_x, w_y) \in E(g_1)$ , we have  $x - y \equiv 1 \pmod{t}$ , and thus  $(x+1) - (y+1) \equiv 1 \pmod{t}$ , which implies  $(f(w_x), f(w_y)) = (w_{x+1}, w_{y+1}) \in E(g_2)$ . Thus, the corresponding vertices have the same number of non-neighbors within the induced subgraph.

- **Step 4: Repetition.** Increase  $i$  by one and go to Step 2.

Clearly, the invariants  $o_i < o_{i+1}$  and  $|V(g'_i)| = |V(g'_{i+1})| = |V(g_{sol})|$  are kept during the above construction. Therefore, it can terminate and find the largest  $k$ -defective clique  $g'$  in  $g$  that contains  $g_{opt}$  and has more vertices than  $g_{opt}$ , i.e.,  $V(g_{opt}) \subset V(g')$ .  $\square$

**Time complexity of IRSolver.** The time complexity of IRSolver is bounded by  $O(|S||C| + |C|^2)$ . To support the greedy selection efficiently, we employ bucket queues (a form of linear heap) to dynamically maintain, for each vertex  $v$  in the current candidate set, the key  $k(v) = 3 \cdot \bar{d}_{S_{opt}}(v) + \bar{d}_{C_{temp}}(v)$ . This key exactly encodes the lexicographic order first by  $\bar{d}_{S_{opt}}(v)$  and then by  $\bar{d}_{C_{temp}}(v)$ , since  $\bar{d}_{C_{temp}}(v) \leq 2$ . Moreover, since  $\bar{d}_{S_{opt}}(v) \leq |S| + |C|$ , we have  $k(v) \in [0, 3(|S| + |C|)]$ . Accordingly, we use an array of size  $3(|S| + |C|) + 1$ , where the  $i$ -th bucket stores a linked list of vertices whose key equals  $i$ . The initialization of IRSolver requires computing  $\bar{d}_{S_{opt}}(v)$  and  $\bar{d}_{C_{temp}}(v)$  for all vertices in  $C$ , which takes  $O(|S||C| + |C|^2)$  time. During the greedy process, the arg min operation for selecting  $v^*$  over  $\Gamma_{min}$  is supported directly by the bucket structure, rather than by recomputing the minimum from scratch in each iteration. In particular, the minimum non-empty bucket exactly contains the vertices in  $\Gamma_{min}$  that are minimal under the above lexicographic order. Since the maintained key is non-decreasing during the process, the pointer to the minimum non-empty bucket moves forward at most  $O(|S| + |C|)$  times in total. In addition, deleting the selected vertex  $v^*$  takes  $O(1)$  time, and updating all affected

vertices also takes  $O(1)$  time per iteration, since each vertex in  $C$  has at most two non-neighbors within  $C$ . Hence, after initialization, the total maintenance cost of the bucket structure is  $O(|S| + |C|)$ . Therefore, the overall time complexity of IRSolver is dominated by the initialization step and remains  $O(|S||C| + |C|^2)$ .

### 4.3 BBRes: New Branching Strategy

**Motivation.** Consider the branching procedure at a branch  $(g, S, C)$ . Recent studies have shown that the performance of the proposed method depends on the pivot selection strategy used during branching. We first briefly revisit the existing pivot selection strategy [12]. Specifically, it selects from  $C$  the vertex that has at least one non-neighbor in  $S$  as the pivot  $v_p$ ; if no such vertex exists, it selects from  $C$  an arbitrary vertex as the pivot  $v_p$ . We observe that the existing branching strategy [12] could select those vertices that have at most two non-neighbors in  $C$  as the pivot. We note that those vertices can be skipped when selecting the pivot, and if no pivot can be chosen from  $C$ , the branch  $(g, S, C)$  falls in the MissingTwoDeg problem instances and can be solved by IRSolver. Therefore, the existing branching strategy is not suitable for our framework as it will create some redundant branches.

**Our branching strategy.** Motivated by the above, we propose the following branching strategy called **BS-three**.

- **BS-three.** Given a branch  $(g, S, C)$ , the pivot  $v_p$  is selected as the vertex in  $C$  that (1) has at least *three* non-neighbors in  $C$  and (2) has at least one non-neighbor in  $S$ ; if no such vertex exists, the pivot  $v_p$  is chosen as an arbitrary vertex in  $C$  that has at least *three* non-neighbors in  $C$ .

We remark that if **BS-three** cannot find a pivot in  $C$ , the branch becomes a MissingTwoDeg problem and can be solved by IRSolver. In addition, **BS-three** cannot be applied to the previous BB framework [11, 12, 17] since they cannot handle the MissingTwoDeg problems (when no pivot can be found in  $C$ ).

**Time complexity of BBRes.** With the proposed **BS-three**, our BBRes will generate  $O(\lambda_k^{\delta\Delta})$  branches when  $|V(g^*)| \geq k + 1$ , and generate  $O(\lambda_k^n)$  branches otherwise in the worst case. Here,  $\delta$  is the degeneracy of  $G$ ,  $\Delta$  is the maximum degree of  $G$ , and  $\lambda_k$  is the largest real root of the equation  $x^{k+4} - 2x^{k+3} + x^3 - x + 1 = 0$  (based on the analytical method in [22]). For example,  $\lambda_k = 1.381, 1.705, 1.867$  when  $k = 1, 2, 3$ . Thus, BBRes has the worst-case time complexity of  $O^*(\lambda_k^{\delta\Delta})$  when  $|V(g^*)| \geq k + 1$ , and of  $O(\lambda_k^n)$  otherwise, where  $O^*$  suppresses the polynomials, as summarized in Lemma 4.3. Recall that the state-of-the-art method following Algorithm 1 has the time complexity of  $O^*(\gamma_k^n)$  where  $\gamma_k$  is the largest real root of  $x^{k+3} - 2x^{k+2} + x^2 - x + 1 = 0$ , e.g.,  $\gamma_k = 1.466, 1.755, 1.889$  for  $k = 1, 2, 3$ . We remark that  $\lambda_k$  is *strictly smaller* than  $\gamma_k$  for  $k \geq 1$ . A formal proof is provided in Appendix A of our technical report [50]; see the proof of Lemma 4.3. Moreover, for  $k = 1, 3, 5, 10, 15, 20$ , the corresponding values of  $\gamma_k$  are 1.465, 1.8885, 1.9750, 1.9993, 1.99998, and 1.9999993, while the corresponding values of  $\lambda_k$  are 1.3803, 1.8668, 1.9706, 1.9991, 1.99997, and 1.9999992.

**LEMMA 4.3.** *Given a graph  $G$  and a integer  $k$ , BBRes runs in  $O(n(\Delta\delta)^2\lambda_k^{\delta\Delta})$  when the largest  $k$ -defective clique in  $G$  is of size larger than  $k + 1$  (i.e.,  $|V(g^*)| \geq k + 2$ ), and runs in  $O(n^2\lambda_k^n)$ , where*

*$\delta$  is the degeneracy of  $G$ ,  $\Delta$  is the maximum degree of  $G$ , and  $\lambda_k$  is the largest real root of the equation  $x^{k+4} - 2x^{k+3} + x^3 - x + 1 = 0$ .*

**PROOF.** We note that each recursion of BBRes runs in polynomial time  $O(|S||C| + |C|^2)$ , which is dominated by IRSolver (details are discussed in Section 4.2). When  $|V(g^*)| \geq k + 2$ , we can utilize the diameter-two property, then the size of  $G_{v_i}$  can be bounded by  $\Delta\delta$  [12] and thus  $|C| \leq \Delta\delta$ ; otherwise, we have  $|C| \leq n$ . We then analyze the number of recursions (or branches) and put the details in Appendix A of our technical report [50].  $\square$

## 5 DOUBLE-COLORING UPPER BOUND

The upper bound estimation is a crucial component in algorithms for the maximum  $k$ -defective problem, as it directly impacts the efficiency and effectiveness of the BB search process. Existing state-of-the-art methods typically rely on graph coloring techniques [11, 12, 17]. By assigning colors to vertices in the candidate set, these methods partition the candidate set into several independent sets, thereby facilitating the computation of upper bounds. However, this classic approach makes a simplifying assumption: it treats every pair of vertices in different color classes as if they must be connected by an edge. While this assumption simplifies the analysis, it often leads to loose upper bounds, especially in graphs where the connectivity between color classes is sparse.

Recognizing this limitation, we propose to advance upper bound computation by introducing a novel *double-coloring* strategy in this section. Rather than relying on a single coloring, we assign each vertex a pair of colors, derived from two independent coloring processes. This dual perspective enables us to capture more refined relationships between vertices. For example, if two vertices fall into different color classes in the first coloring but share the same color in the second, we gain additional insight: despite what the first coloring suggests, these vertices actually belong to an independent set under the second coloring. To illustrate, consider the example in Figure 2, which contrasts the granularity of structural information captured by the two schemes. As shown in Figure 2(a), single coloring assigns identical colors to non-adjacent vertices (e.g., the blue nodes), implicitly treating them as a clique and thus identifying only 2 missing edges. In contrast, Figure 2(b) demonstrates that double coloring distinguishes these vertices via unique color pairs, successfully exposing all 4 missing edges (indicated by dashed lines). In the following, we first present the computation of the double-coloring upper bound (Section 5.1) and then prove its correctness (Section 5.2).

### 5.1 Double-Coloring Upper Bound Computation

**Additional notations.** We now leverage the richer structural information revealed by double coloring. Specifically, for each vertex  $v$  in the candidate set  $C$ , we assign a pair of colors,  $(col_1(v), col_2(v))$ , where  $col_1(\cdot)$  and  $col_2(\cdot)$  are two different vertex colorings. Both colorings respect the graph's edges: for any edge  $(u, v)$  in  $G$ , we ensure  $col_1(u) \neq col_1(v)$  and  $col_2(u) \neq col_2(v)$ . We refer to each ordered pair  $(col_1(v), col_2(v))$  as the *color pair* of  $v$ . The specific procedures for generating these colorings will be introduced later.

We further introduce a *double-coloring indicator* function  $I_G(u, v)$ , which is defined as 1 if two distinct vertices  $u$  and  $v$  share the same color in either of the two colorings, and 0 otherwise. In other words,

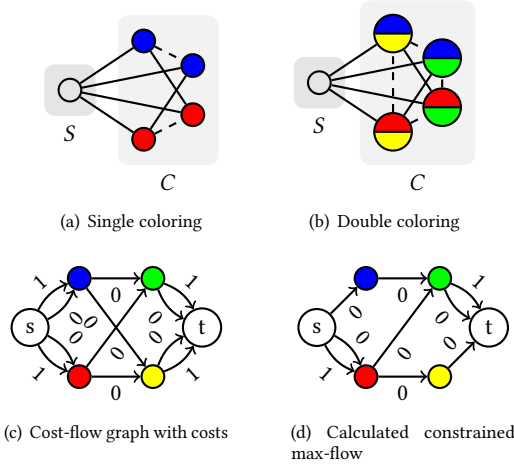


**Figure 2: Comparison of coloring strategies and upper bound computation ( $k = 2$ ).** (a) **Single coloring misses non-edges.** (b) **Double coloring reveals them via unique color pairs.** (c) **Cost-flow network with edge costs.** (d) **Max flow of 3 with cost 2 (flow shown on edges), yielding  $UB = |S| + 3$ .**

$I_G(u, v)$  indicates whether  $u$  and  $v$  are placed in the same color class in at least one of the two independently obtained colorings. It is easy to observe that, for any induced subgraph  $g$ , the quantity  $\frac{1}{2} \sum_{u,v \in V(g)} I_g(u, v)$  provides a lower bound on the number of missing edges in  $g$ , i.e.,  $\frac{1}{2} \sum_{u,v \in V(g)} I_g(u, v) \leq |\bar{E}(g)|$ , where  $|\bar{E}(g)|$  is the number of non-edges in  $g$ . This is because  $I_g(u, v) = 1$  implies that  $u$  and  $v$  are not adjacent in  $g$ , but the converse may not hold.

**Overview of double-coloring upper-bound computation.** Given a branch  $(g, S, C)$ , our goal is to leverage the structural information provided by double coloring to select a subset  $D$  of the candidate set  $C$  such that the size of the maximum  $k$ -defective clique  $g'$  in this branch (i.e.,  $S \subseteq V(g') \subseteq S \cup C$ ) is at most  $|D| + |S|$ . In other words,  $|D| + |S|$  serves as our double-coloring upper bound, denoted as **UB-Double**. However, this is a non-trivial task, as incorporating information from two colorings introduces additional dependencies that must be carefully managed.

To compute **UB-Double**, we construct a corresponding *cost-flow graph* instance based on the current branch  $(g, S, C)$  and employ the classic *constrained maximum flow algorithm* [2, 19, 25]. In this setting, the objective is to maximize the flow (which corresponds to the number of vertices added to  $S$ ), while ensuring that the total cost (which corresponds to the number of non-edges) does not exceed a threshold, say  $k - |\bar{E}(S)|$ . After obtaining the maximum flow in the cost-flow graph, we show that it corresponds to a subset  $D$  of the candidate set  $C$ . Our **UB-Double** is then defined as  $|S| + |D|$ .

In the following, we present our double-coloring upper bound method **UB-Double** based on the cost-flow graph and constrained maximum flow computation.

**Double-coloring upper bound via cost-flow graph.** Given a branch  $(g, S, C)$ , we aim to compute a tight upper bound on the size of the maximum  $k$ -defective clique by leveraging the structural information provided by double coloring. To this end, we

first assign two colorings to the vertices in  $C$ : the first coloring,  $col_1(\cdot)$ , is assigned greedily according to the degeneracy ordering; the second coloring,  $col_2(\cdot)$ , is assigned arbitrarily<sup>1</sup>, with the additional constraint that no two vertices share the same color pair, i.e.,  $(col_1(u), col_2(u)) = (col_1(v), col_2(v))$  only if  $u = v$ .

Based on the assigned colorings, we construct the *cost-flow (directed) graph*  $g^c = (V^c, E^c)$ . First, we define the vertex set  $V^c$ . Let  $V'_1 = \{col_1(v) \mid v \in C\}$  and  $V'_2 = \{col_2(v) \mid v \in C\}$  be the sets of colors derived from the two coloring processes. The vertex set is then defined as  $V^c = \{s\} \cup \{t\} \cup V'_1 \cup V'_2$ . As in Figure 2(c) (with  $k = 2$ ), this vertex set forms a layered topology:  $V'_1$  and  $V'_2$  constitute two distinct layers bridging the source  $s$  and the sink  $t$ . Next, we construct the edge set  $E^c$  to encode the vertex selection and the  $k$ -defective constraint. The construction rules are as follows:

- **Source edges:** For each vertex  $u \in V'_1$ , we add  $k + 1$  parallel edges from  $s$  to  $u$ , denoted as  $(s, u)$ , with capacity 1 and costs  $0, 1, \dots, k$ , respectively. These edges distribute the allowable missing-edge budget.
- **Sink edges:** Similarly, for each vertex  $v \in V'_2$ , we add  $k + 1$  parallel edges from  $v$  to  $t$ , with capacity 1 and costs  $0, 1, \dots, k$ , respectively.
- **Internal edges:** For each vertex  $x \in C$  in the original graph, let  $u = col_1(x)$  and  $v = col_2(x)$ . We add a directed edge  $(u, v)$  connecting the two color layers, with capacity 1 and cost  $\bar{d}_S(x)$ . This edge represents the potential selection of vertex  $x$ .
- **Reverse edges:** For every edge constructed above, we add a corresponding reverse edge with capacity 0 and the negation of the original cost.

Given the constructed cost-flow graph instance  $g^c = (V^c, E^c)$ , we solve the *constrained maximum flow problem* [3, 19, 25], which aims to maximize the flow such that the total cost does not exceed  $k - |\bar{E}(S)|$ . The value of the maximum flow obtained from this graph corresponds precisely to the size of the subset  $D \subseteq C$  that can be added to  $S$  while satisfying the  $k$ -defective constraint. Our double-coloring upper bound, denoted as **UB-Double**, is then defined as  $|S| + |D|$ . As an illustration, Figure 2(d) presents the computation result for the running example (where  $k = 2$ ). The algorithm identifies a valid flow of value 3 with a total cost of 2. Since the cost is within the budget, this implies a valid subset size  $|D| = 3$ , yielding a refined upper bound of  $|S| + 3 = 4$ . Furthermore, there exists a direct mapping between flows in  $g^c$  and the subset  $D$ . Specifically, if a unit of flow in the maximum flow solution passes through an edge  $(u, v)$  with  $u \in V'_1$  and  $v \in V'_2$ , we include the corresponding vertex in the set  $D$ . To formalize this mapping, let  $p : v \rightarrow (col_1(v), col_2(v))$  denote the correspondence from  $g$  to  $g^c$ , where  $v \in C$  and  $(col_1(v), col_2(v)) \in E(g^c)$ .

## 5.2 Correctness and Complexity of Our Double-Coloring Upper Bound

We now establish the correctness proof and analyze the time complexity of our double-coloring upper bound **UB-Double**.

<sup>1</sup>In our current implementation, the vertices in  $C$  are processed according to their current memory order. As shown in Appendix D of the technical report, this strategy achieves the best overall performance among the tested second-coloring orders. Designing more effective second-coloring orders remains an interesting future work.

**Correctness proof.** We first establish the correctness of our proposed **UB-Double**. Specifically, given any branch  $(g, S, C)$  and its corresponding cost-flow graph instance  $g^c$ , our goal is to show that for any  $k$ -defective clique  $g'$  in this branch (where  $S \subseteq V(g') \subseteq S \cup C$ ), the size of  $g'$  cannot exceed  $|S| + |D|$ , where  $D \subseteq C$  is the subset selected by our algorithm.

The central idea of our proof is to relate the number of missing edges in the induced subgraph  $G[S \cup D]$  to the cost of the constrained maximum flow in the cost-flow graph. Specifically, we observe that the total number of missing edges in  $G[S \cup D]$  is equal to the sum of three terms:  $|\bar{E}(S \cup D)| = |\bar{E}(S)| + |\bar{E}(D)| + \sum_{u \in D} \bar{d}_S(u)$ . Our proof proceeds in two main steps:

- First, as shown in Lemma 5.1, for any subset  $D$ , the double-coloring indicator provides a lower bound on the number of missing edges within  $D$ , i.e.,  $\frac{1}{2} \sum_{u,v \in D} I_G(u, v) \leq |\bar{E}(D)|$ .
- Second, as established in Lemma 5.2, the cost  $\text{cost}(g^c)$  computed in the cost-flow graph  $g^c$  exactly matches the sum of the double-coloring indicator over  $D$  and the number of missing edges between  $S$  and  $D$ , i.e.,  $\text{cost}(g^c) = \frac{1}{2} \sum_{u,v \in D} I_G(u, v) + \sum_{u \in D} \bar{d}_S(u)$ . Putting these observations together, we see that if the total cost in the cost-flow graph does not exceed  $k - \bar{E}(S)$ , then the total number of missing edges in any  $k$ -defective clique formed by  $S$  and  $D$  is at most  $k$ . This means that any  $k$ -defective clique can be represented as a feasible flow in our construction. Since our constrained maximum flow algorithm finds the largest possible  $D$  under the cost constraint, the solution  $|S| + |D|$  serves as a valid upper bound on the size of any  $k$ -defective clique containing  $S$  in this branch. Thus, our **UB-Double** produces a correct upper bound.

We now provide the detailed proof. We begin by presenting a lemma that follows directly from the definition of the double-coloring indicator.

LEMMA 5.1. *Given a graph  $G = (V, E)$  and any subset  $D \subseteq V$ , we have  $\frac{1}{2} \sum_{u,v \in D} I_G(u, v) \leq |\bar{E}(D)|$ .*

We are ready to give the following result, where the complete derivation is provided in Appendix A of our technical report [50].

LEMMA 5.2. *Given a constructed cost-flow graph  $g^c$  and its constrained maximum flow, there exists a corresponding vertex set  $D$  in  $C$  such that  $\text{cost}(g^c) = \frac{1}{2} \sum_{u,v \in D} I_G(u, v) + \sum_{u \in D} \bar{d}_S(u)$ .*

**Effectiveness of UB-Double over UB-Single.** We first briefly review the standard single-coloring-based upper bound, denoted by **UB-Single** [11, 12, 17]. Given a branch  $(g, S, C)$ , previous methods compute an upper bound by assigning a single coloring  $\text{col}(\cdot)$  to the vertices in the candidate set  $C$  according to the degeneracy ordering. This is exactly the first coloring step used in our double-coloring method **UB-Double**. The resulting bound is denoted by **UB-Single**. We further show that our **UB-Double** is always no larger than the existing single-coloring-based upper bound **UB-Single** [11, 12, 17], and is therefore always at least as tight.

LEMMA 5.3. *Given an instance  $(g, S, C)$ ,  $\text{UB-Double} \leq \text{UB-Single}$ .*

The proof is provided in Appendix A of our technical report [50].

**Time complexity of UB-Double.** Compared with the standard single-coloring-based upper bound **UB-Single**, computing **UB-Double** incurs additional overhead. Consider a branch  $(g, S, C)$ .

**Table 1: Number of solved instances by the algorithms with a 3-hour limit (best performers are highlighted in bold).**

|          | Real-world graphs |      |     |      |            |
|----------|-------------------|------|-----|------|------------|
|          | BBRes             | kDC2 | MDC | DnBk | WODC       |
| $k = 1$  | <b>139</b>        | 137  | 138 | 135  | 138        |
| $k = 3$  | <b>139</b>        | 136  | 136 | 135  | <b>139</b> |
| $k = 5$  | <b>139</b>        | 136  | 131 | 134  | <b>139</b> |
| $k = 10$ | <b>135</b>        | 128  | 127 | 124  | 134        |
| $k = 15$ | <b>132</b>        | 126  | 121 | 119  | 130        |
| $k = 20$ | <b>129</b>        | 114  | 113 | 111  | 127        |

**UB-Double** first colors the vertices in  $C$  twice: the first coloring requires  $O(|E(g[C])|)$  time [8], and the second coloring requires  $O(|C| \times c) = O(|C|^2)$  time, where  $c$  is the number of colors for  $C$  and  $c \leq |C|$ . Constructing the corresponding cost-flow graph takes  $O(ck) = O(|C|k)$  time. Finally, we apply a classical maximum flow algorithm [2, 19, 25] to solve the problem, which runs in  $O(k|C| \log |C|)$  time. Thus, the overall time complexity of **UB-Double** is  $O(k|C| \log |C| + |C|^2)$ . By comparison, computing **UB-Single** needs  $O(|C| + |E(g[C])|)$  time [11, 12, 17], which is smaller. Nevertheless, in our setting,  $|C|$  is bounded by the graph degeneracy  $\delta$ , which is typically small on real-world graphs. Hence, computing **UB-double** remains practically efficient.

## 6 EXPERIMENTS

In this section, we conduct experiments to evaluate the performance of our proposed algorithm BBRes against four baselines.

- kDC2 <sup>2</sup>: the existing algorithm proposed in [12].
- MDC <sup>3</sup>: the existing method proposed in [17].
- DnBK <sup>4</sup>: the existing algorithm proposed in [34], augmented with the same Stage II techniques (Algorithm 1) as kDC2 and MDC, for solving the maximum  $k$ -defective clique problem without the size constraint.
- WODC <sup>5</sup>: the state-of-the-art method proposed in [28].

We note that prior algorithms, including MADEC<sup>+</sup> [14], KDBB [23], KD-Club [29], and kDC [11], run significantly slower than the above baselines proposed recently and thus are omitted for the comparison. All algorithms are implemented in C++, compiled with -O3, and run on a machine with an Intel CPU @ 2.60GHz and 256GB main memory. Following previous studies [12, 17, 34], we choose  $k$  from  $\{1, 3, 5, 10, 15, 20\}$  and set the time limit as 3 hours (i.e., 10,800s) for a fair comparison. We evaluate the algorithms on a widely adopted collection <sup>6</sup> of **139** real-world graphs [12, 17, 34] containing a total of  $5.87 \times 10^7$  vertices. Our source code can be found in [50].

### 6.1 Comparison with Baselines

**Number of solved instances.** Table 1 summarizes the number of instances solved by each algorithm within the 3-hour cutoff. First, we observe a general downward trend in the number of solved instances as  $k$  increases, confirming the exponential growth in computational complexity for larger  $k$ . Despite this challenge, BBRes consistently demonstrates superior performance, achieving the

<sup>2</sup><https://lijunchang.github.io/kDC-two/>

<sup>3</sup><https://github.com/dawhc/MaximumDefectiveClique/>

<sup>4</sup><https://github.com/cy-Luo000/Maximum-k-Defective-Clique/>

<sup>5</sup><https://github.com/SNUCSE-CTA/WODC/>

<sup>6</sup><http://lcs.ios.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz>

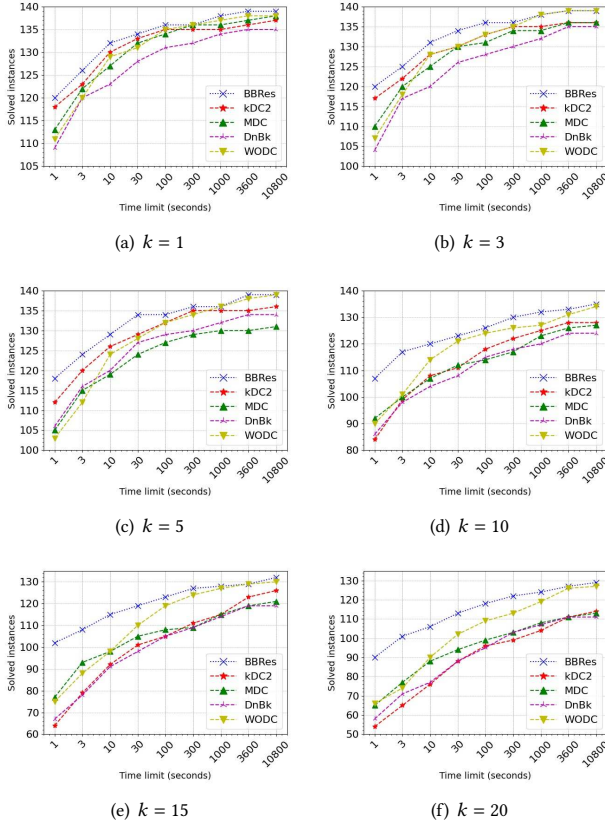


**Figure 3: Number of solved instances with varying time limits**

highest coverage across all settings. Significantly, the performance gap between BBRes and the competitors widens as  $k$  increases. For example, at  $k = 20$ , BBRes solves 129 instances, outperforming the runner-up (WODC) by 2 instances and significantly surpassing kDC2 by 15 instances. We attribute this advantage to the effective synergy between our diameter-2 partitioning and the proposed double-coloring upper bound. Real-world graphs typically exhibit a “globally sparse, locally dense” topology. While diameter-2 partitioning decomposes the globally sparse structure into smaller sub-instances, these sub-instances often retain high local density. It is in these dense regions that baseline methods falter, as their bounds (e.g., single-coloring) become loose, leading to redundant branches. In contrast, our double-coloring strategy captures finer-grained structural conflicts, thereby tightening the bounds and effectively pruning the search space in these dense substructures.

We extend our analysis to the time-efficiency trade-off by plotting the number of solved instances against the runtime budget (ranging from 1 second to 3 hours) in Figure 3. Across all tested values of  $k$ , BBRes maintains a dominant position, indicated by its curve consistently lying above those of the competitors (kDC2, MDC, DnBk, and WODC). Two key observations highlight the efficacy of BBRes: First, BBRes excels in rapid convergence. For hard instances where  $k = 20$ , BBRes quickly solves approximately 90 instances in the first second, whereas the baselines struggle to solve 65. Even

with a generous 3-hour limit, BBRes reaches a total of 129 solved instances, surpassing the best baseline (WODC, 127 instances). Second, our method demonstrates superior robustness to problem difficulty. While the performance of baseline algorithms drops sharply as  $k$  increases (reflecting the exponential growth in search space), BBRes sustains a high completion rate. This suggests that our double-coloring bound effectively prunes the search space, mitigating the impact of increasing  $k$ .

**Runtime efficiency on representative benchmarks.** We evaluate the runtime performance of BBRes against four baselines, namely kDC2, MDC, DnBk, and WODC, on representative graphs with  $k \in \{5, 10, 15, 20\}$ . The dataset consists of the 30 largest graphs from the real-world collection, excluding instances where all algorithms exceeded the 3-hour time limit at  $k = 20$ . Table 2 reports the detailed running times. The results demonstrate that BBRes consistently delivers superior performance across all tested values of  $k$ , except for a few trivial instances. Notably, on the soc-flixster graph with  $k = 20$ , BBRes achieves a remarkable 53 $\times$  speedup over the best-performing baseline (WODC, 89.1s vs. 1.7s). In addition, we can observe that BBRes achieves at least a 2 $\times$  speedup over the best competing method on 26.7%, 20.0%, 40.0%, and 53.3% of the test cases for  $k = 5, 10, 15$ , and 20, respectively.

Furthermore, BBRes exhibits significant scalability and robustness with respect to increasing  $k$ . In contrast to the baselines, whose runtimes often degrade exponentially as  $k$  grows, BBRes remains highly efficient. For instance, on socfb-A-anon, as  $k$  increases from 10 to 20, the runtime of BBRes grows moderately from 7.8s to 17.1s. In stark contrast, the competitors suffer from severe performance degradation at  $k = 20$ : MDC, kDC2, and DnBk require 140.9s, 988.4s, and 1304.3s, respectively. Even the runner-up, WODC, is significantly slower, taking 57.3s, which is more than 3 $\times$  that of BBRes. These results strongly validate the practical efficiency of our proposed algorithm in handling difficult constraints. Finally, we note that BBRes runs slightly slower than the baselines on some instances in Table 2. This mainly occurs on relatively easy instances where all methods terminate within a fraction of a second. In such cases, the computational overhead of BBRes’s advanced techniques (e.g., **UB-Double**) may outweigh their pruning benefits.

## 6.2 Ablation Studies

To rigorously assess the individual contributions of our proposed techniques, we conduct an ablation study by comparing BBRes against two distinct variants. Specifically, we define -FlowUB as the variant that replaces our double-coloring upper bound (Section 5) with a standard single-coloring bound, thereby isolating the impact of our novel bounding strategy. Meanwhile, we define -Branch as the variant that disables our specific branching and early termination strategies (Section 4). We note that -Branch adopts the state-of-the-art branching strategy proposed in [17] for fair comparison, and it differs with MDC mainly in the upper bound. This comparison allows us to quantify the performance gain attributed solely to our optimized branching and early termination strategies.

Table 3 details the runtime performance of these methods across 30 representative benchmark graphs. The results consistently demonstrate that the full BBRes algorithm achieves superior performance compared to both variants in the vast majority of cases.

**Table 2: Running time (in seconds) on 30 representative real-world benchmark graphs. ‘-’ indicates timeout. The best performer is highlighted in bold; specifically, if the running time is within 10% of the fastest time, it is considered as the best.**

| Graphs            | <i>n</i> | <i>m</i> | <i>k</i> = 5   |              |         |              |              | <i>k</i> = 10  |              |          |              |               | <i>k</i> = 15  |              |          |              |               | <i>k</i> = 20   |              |         |              |                |
|-------------------|----------|----------|----------------|--------------|---------|--------------|--------------|----------------|--------------|----------|--------------|---------------|----------------|--------------|----------|--------------|---------------|-----------------|--------------|---------|--------------|----------------|
|                   |          |          | BBRes          | kDC2         | MDC     | DnBk         | WDC          | BBRes          | kDC2         | MDC      | DnBk         | WDC           | BBRes          | kDC2         | MDC      | DnBk         | WDC           | BBRes           | kDC2         | MDC     | DnBk         | WDC            |
| socfb-A-anon      | 3M       | 23M      | 6.513          | 6.65         | 19.343  | 21.89        | <b>6.645</b> | 7.793          | 37.453       | 32.101   | 48.902       | 8.919         | <b>10.689</b>  | 289.987      | 52.543   | 305.076      | 18.556        | <b>17.136</b>   | 988.404      | 140.895 | 1304.348     | 57.337         |
| soc-orkut         | 2M       | 106M     | <b>104.263</b> | 157.448      | 137.127 | 390.314      | 405.808      | <b>112.963</b> | 576.496      | 188.372  | 523.604      | 455.723       | <b>129.552</b> | 8249.529     | -        | 1353.479     | 573.453       | <b>146.826</b>  | -            | -       | -            | 892.649        |
| socfb-B-anon      | 2M       | 20M      | 10.165         | 8.984        | 18.302  | 21.135       | <b>6.883</b> | 12.272         | 58.884       | 21.368   | 66.563       | 7.622         | 25.553         | 1737.898     | 45.803   | 778.94       | <b>22.41</b>  | <b>19.994</b>   | 8811.373     | 378.424 | -            | 79.273         |
| soc-flixster      | 2M       | 7M       | <b>0.521</b>   | 4.136        | 2.74    | 2.461        | 1.189        | <b>0.594</b>   | 38.122       | 3.875    | 8.435        | 1.871         | <b>0.788</b>   | 1199.894     | 19.231   | 84.602       | 10.753        | <b>1.677</b>    | -            | 294.896 | 1785.488     | 89.086         |
| web-wikipedia2009 | 1M       | 4M       | <b>0.84</b>    | <b>0.783</b> | 2.083   | 1.289        | 1.872        | 0.954          | <b>0.846</b> | 2.779    | 2.551        | 2.542         | <b>0.973</b>   | <b>0.891</b> | 2.889    | 49.022       | 2.726         | <b>1.66</b>     | <b>1.574</b> | 3.085   | 16.88        | 2.596          |
| tech-as-skitter   | 1M       | 11M      | 0.707          | <b>0.593</b> | 1.877   | 0.982        | 1.288        | <b>0.755</b>   | <b>0.702</b> | 1.972    | 2.173        | 1.26          | <b>0.886</b>   | 1.8          | 2.058    | 3.624        | 1.6           | <b>1.04</b>     | 21.869       | 2.804   | 10.765       | 2.498          |
| soc-pokec         | 1M       | 22M      | 5.996          | <b>4.557</b> | 11.098  | 16.999       | 7.88         | 6.588          | <b>5.188</b> | 14.35    | 17.144       | 7.634         | <b>9.899</b>   | <b>9.381</b> | 18.949   | 101.537      | <b>9.261</b>  | <b>13.476</b>   | 58.022       | 23.248  | 430.095      | 24.827         |
| soc-lastfm        | 1M       | 4M       | 3.392          | 105.634      | 88.744  | 104.585      | <b>2.992</b> | <b>13.906</b>  | 2826.062     | 2098.176 | -            | <b>14.784</b> | <b>64.639</b>  | -            | -        | -            | <b>70.443</b> | <b>529.874</b>  | -            | -       | -            | <b>493.475</b> |
| soc-youtube-snap  | 1M       | 2M       | <b>0.677</b>   | 3.844        | 2.42    | 3.009        | 1.247        | <b>1.705</b>   | 56.95        | 28.339   | 31.979       | 3.735         | <b>13.467</b>  | 1341.397     | 921.233  | 938.395      | 37.626        | <b>193.191</b>  | -            | -       | -            | 497.69         |
| ca-hollywood-2009 | 1M       | 56M      | <b>0.856</b>   | <b>0.825</b> | 3.263   | <b>0.795</b> | 3.36         | 0.854          | <b>0.818</b> | 3.315    | <b>0.774</b> | 3.35          | <b>0.855</b>   | <b>0.814</b> | 3.364    | <b>0.782</b> | 3.174         | <b>0.867</b>    | <b>0.835</b> | 3.401   | <b>0.795</b> | 3.15           |
| sc-ldoor          | 952K     | 20M      | <b>21.676</b>  | 64.183       | 47.185  | -            | 494.426      | <b>143.497</b> | 1669.188     | 894.263  | -            | 2290.9        | <b>361.145</b> | 3532.593     | 3093.435 | -            | 4353.16       | <b>4114.424</b> | -            | -       | -            | -              |
| soc-digg          | 770K     | 5M       | 25.555         | 38.424       | 43.787  | 1144.44      | 47.122       | <b>33.257</b>  | 177.117      | 395.428  | 1885.92      | 61.119        | <b>41.287</b>  | 8896.453     | 852.619  | -            | 266.135       | <b>64.569</b>   | -            | 9918.25 | -            | 3046.37        |
| ca-coauthors-dblp | 540K     | 15M      | 0.24           | <b>0.205</b> | 0.815   | <b>0.224</b> | 0.839        | 0.319          | <b>0.245</b> | 0.845    | 0.315        | 3.514         | 0.331          | <b>0.258</b> | 0.874    | 0.381        | 5.966         | 0.334           | <b>0.269</b> | 0.893   | 0.453        | 5.978          |
| soc-delicious     | 536K     | 1M       | 0.075          | <b>0.068</b> | 0.437   | 0.151        | 0.169        | <b>0.113</b>   | 0.135        | 0.417    | 0.295        | 0.301         | <b>0.179</b>   | 0.391        | 0.587    | 0.686        | 0.474         | <b>0.232</b>    | 4.095        | 1.537   | 2.157        | 1.037          |
| web-it-2004       | 509K     | 7M       | 0.163          | <b>0.14</b>  | 0.42    | 0.4          | 0.159        | 0.231          | <b>0.202</b> | 0.579    | 0.704        | 16.653        | <b>0.237</b>   | <b>0.214</b> | 0.603    | 0.709        | 16.907        | <b>0.243</b>    | <b>0.225</b> | 0.607   | 0.736        | 16.821         |
| soc-youtube       | 495K     | 1M       | <b>0.418</b>   | 1.619        | 1.28    | 1.953        | 1.123        | <b>1.082</b>   | 31.709       | 16.115   | 17.539       | 4.14          | <b>7.795</b>   | 758.326      | -        | 509.977      | 41.827        | <b>102.463</b>  | -            | -       | -            | 674.757        |
| sc-msdoor         | 415K     | 9M       | <b>10.726</b>  | 31.332       | 24.22   | 1343.813     | 241.336      | <b>81.904</b>  | 874.654      | 647.598  | 2487.434     | 1164.13       | <b>199.684</b> | 1897.013     | 2263.7   | -            | 2211.78       | <b>2360.887</b> | -            | -       | -            | -              |
| ca-MathSciNet     | 332K     | 820K     | 0.044          | <b>0.032</b> | 0.243   | 0.036        | 0.103        | 0.045          | <b>0.032</b> | 0.283    | 0.049        | 0.117         | 0.051          | <b>0.038</b> | 0.309    | 0.045        | 0.148         | 0.098           | <b>0.08</b>  | 0.675   | 0.444        | 0.772          |
| ca-dblp-2012      | 317K     | 1M       | 0.061          | <b>0.043</b> | 0.246   | <b>0.043</b> | 0.108        | 0.055          | <b>0.045</b> | 0.244    | <b>0.047</b> | 0.096         | 0.071          | <b>0.047</b> | 0.275    | 0.053        | 0.185         | 0.062           | <b>0.037</b> | 0.236   | 0.054        | 0.215          |
| ca-citeseer       | 227K     | 814K     | 0.034          | <b>0.026</b> | 0.175   | 0.029        | 0.07         | 0.03           | <b>0.021</b> | 0.156    | 0.027        | 0.07          | 0.025          | <b>0.021</b> | 0.172    | 0.031        | 0.084         | <b>0.029</b>    | <b>0.028</b> | 0.152   | 0.035        | 0.125          |
| ca-dblp-2010      | 226K     | 716K     | 0.023          | <b>0.018</b> | 0.138   | 0.021        | 0.067        | 0.022          | <b>0.019</b> | 0.136    | <b>0.02</b>  | 0.058         | <b>0.023</b>   | <b>0.025</b> | 0.147    | <b>0.023</b> | 0.103         | 0.025           | <b>0.019</b> | 0.142   | 0.028        | 0.133          |
| sc-pwtik          | 217K     | 5M       | <b>1.679</b>   | 2.065        | 3.675   | 420.808      | 72.627       | <b>5.801</b>   | 18.982       | 12.244   | 589.325      | 136.318       | <b>32.956</b>  | 1544.762     | 180.301  | 1010.477     | 787.769       | <b>133.684</b>  | 1655.622     | 451.792 | 1657.136     | 17.799         |
| soc-gowalla       | 196K     | 950K     | <b>0.182</b>   | 0.219        | 0.314   | 0.241        | <b>0.176</b> | <b>0.357</b>   | 1.409        | 0.406    | 0.489        | <b>0.352</b>  | <b>0.748</b>   | 67.38        | 0.831    | 1.937        | <b>0.793</b>  | 6.406           | 2115.989     | 3.487   | 17.82        | 2.701          |
| tech-RL-caida     | 190K     | 607K     | <b>0.14</b>    | <b>0.131</b> | 0.176   | 0.178        | 0.17         | <b>0.09</b>    | 0.179        | 0.188    | 0.361        | 0.339         | <b>0.122</b>   | 1.873        | 1.075    | 1.282        | 0.833         | <b>0.294</b>    | 14.532       | -       | 10.435       | 2.249          |
| sc-shipsec5       | 179K     | 2M       | <b>0.176</b>   | <b>0.161</b> | 0.44    | 0.693        | 1.665        | <b>0.258</b>   | 0.307        | 0.68     | 1.826        | 2.867         | <b>0.884</b>   | 6.494        | 2.288    | 15.372       | 10.85         | <b>2.344</b>    | 14.16        | 6.433   | 107.36       | 25.538         |
| web-arabic-2005   | 163K     | 1M       | <b>0.02</b>    | <b>0.022</b> | 0.112   | 0.023        | 0.088        | <b>0.02</b>    | 0.023        | 0.119    | 0.024        | 0.088         | <b>0.026</b>   | <b>0.027</b> | 0.123    | <b>0.028</b> | 0.145         | <b>0.026</b>    | <b>0.024</b> | 0.121   | <b>0.024</b> | 0.146          |
| soc-douban        | 154K     | 327K     | <b>0.03</b>    | <b>0.029</b> | 0.117   | 0.109        | 0.09         | <b>0.029</b>   | <b>0.029</b> | 4318.654 | 0.14         | 0.234         | 7.509          | 39.523       | -        | <b>7.028</b> | 15.019        | 51.643          | -            | -       | -            | 5.31           |
| sc-shipsec1       | 140K     | 1M       | 0.14           | <b>0.096</b> | 0.216   | 0.13         | 0.337        | <b>0.18</b>    | <b>0.169</b> | 0.624    | 0.927        | 1.812         | <b>0.691</b>   | <b>0.721</b> | 1.677    | 3.125        | 5.079         | <b>1.622</b>    | 6.267        | 28.64   | 34.677       | 17.956         |
| web-uk-2005       | 129K     | 11M      | 0.252          | <b>0.213</b> | 0.472   | 0.65         | 17.792       | <b>0.304</b>   | <b>0.268</b> | 0.575    | 0.839        | 25.944        | <b>0.316</b>   | <b>0.285</b> | 0.595    | 1.002        | 26.144        | <b>0.331</b>    | <b>0.302</b> | 0.616   | 1.088        | 26.286         |
| web-sk-2005       | 121K     | 334K     | <b>0.01</b>    | <b>0.01</b>  | 0.055   | <b>0.011</b> | 0.132        | <b>0.012</b>   | <b>0.011</b> | 0.066    | 0.014        | 0.208         | <b>0.01</b>    | <b>0.011</b> | 0.056    | 0.015        | 0.213         | 0.016           | <b>0.012</b> | 0.067   | 0.016        | 0.217          |

**Table 3: Running time (in seconds) comparison of BBRes and its variants on 30 benchmark graphs.**

| Graphs            | <i>n</i> | <i>m</i> | <i>k</i> = 5 |         |         |         | <i>k</i> = 10 |         |         |          | <i>k</i> = 15 |         |          |          | <i>k</i> = 20 |          |         |          |
|-------------------|----------|----------|--------------|---------|---------|---------|---------------|---------|---------|----------|---------------|---------|----------|----------|---------------|----------|---------|----------|
|                   |          |          | BBRes        | -FlowUB | -Branch | Color   | BBRes         | -FlowUB | -Branch | Color    | BBRes         | -FlowUB | -Branch  | Color    | BBRes         | -FlowUB  | -Branch | Color    |
| socfb-A-anon      | 3M       | 23M      | 6.510        | 6.570   | 5.990   | 3.170   | 7.790         | 7.960   | 7.170   | 32.120   | 10.689        | 11.151  | 9.888    | 261.946  | 17.136        | 18.292   | 16.070  | 921.318  |
| soc-orkut         | 2M       | 106M     | 104.263      | 105.429 | 97.701  | 184.354 | 112.963       | 122.661 | 105.647 | 867.689  | 129.552       | 175.951 | 119.789  | -        | 146.826       | 314.638  | 153.571 | -        |
| socfb-B-anon      | 2M       | 20M      | 10.165       | 10.030  | 10.589  | 10.925  | 12.272        | 12.914  | 13.226  | 61.008   | 25.553        | 22.276  | 22.370   | 1680.934 | 19.994        | 22.902   | 20.760  | 8305.512 |
| soc-flixster      | 2M       | 7M       | 0.521        | 0.990   | 0.538   | 5.636   | 0.594         | 1.670   | 0.616   | 52.162   | 0.788         | 4.255   | 1.056    | 1393.552 | 1.677         | 24.123   | 11.537  | -        |
| web-wikipedia2009 | 1M       | 4M       | 0.840        | 0.842   | 0.837   | 0.905   | 0.954         | 0.985   | 0.971   | 1.187    | 0.973         | 0.972   | 1.013    | 1.217    | 1.660         | 1.564    | 1.685   | 1.831    |
| tech-as-skitter   | 1M       | 11M      | 0.707        | 0.618   | 0.719   | 0.562   | 0.755         | 0.660   | 0.802   | 0.639    | 0.886         | 0.797   | 0.901    | 1.395    | 1.040         | 1.132    | 1.069   | 19.536   |
| soc-pokec         | 1M       | 22M      | 5.996        | 6.228   | 5.890   | 5.457   | 6.588         | 6.962   | 6.500   | 6.650    | 9.899         | 10.418  | 9.763    | 12.063   | 13.476        | 14.673   | 12.953  | 63.050   |
| soc-lastfm        | 1M       | 4M       | 3.392        | 13.851  | 3.475   | 117.415 | 13.906        | 63.229  | 17.442  | 2923.227 | 64.639        | 237.258 | 145.164  | -        | 529.874       | 1439.551 | 834.218 | -        |
| soc-youtube-snap  | 1M       | 2M       | 0.677        | 1.588   | 0.671   | 4.083   | 1.705         | 6.757   | 1.699   | 60.676   | 13.467        | 50.892  | 15.248   | 1407.650 | 193.191       | 464.716  | 222.509 | -        |
| ca-hollywood-2009 | 1M       | 56M      | 0.856        | 0.860   | 0.849   | 0.912   | 0.854         | 0.849   | 0.846   | 0.888    | 0.855         | 0.856   | 0.851    | 0.879    | 0.867         | 0.845    | 0.866   | 0.882    |
| sc-ldoor          | 952K     | 20M      | 21.676       | 23.890  | 21.619  | 46.003  | 143.497       | 125.057 | 173.210 | 1869.012 | 361.145       | 377.997 | 1573.838 | 4111.127 | 4114.424      | 3976.471 | -       | -        |
| soc-digg          | 770K     | 5M       | 25.555       | 29.010  | 25.245  | 39.611  | 33.257        | 41.776  | 33.006  | 210.258  | 41.287        | 65.069  | 41.346   | 9546.118 | 64.569        | 287.370  | 107.880 | -        |
| ca-coauthors-dblp | 540K     | 15M      | 0.240        | 0.228   | 0.234   | 0.236   | 0.319         | 0.303   | 0.314   | 0.398    | 0.331         | 0.322   | 0.327    | 0.460    | 0.334         | 0.325    | 0.326   | 0.477    |
| soc-delicious     | 536K     | 1M       | 0.075        | 0.087   | 0.083   | 0.088   | 0.113         | 0.098   | 0.091   | 0.107    | 0.179         | 0.149   | 0.148    | 0.365    | 0.232         | 0.283    | 0.227   | 3.991    |
| web-it-2004       | 509K     | 7M       | 0.163        | 0.158   | 0.166   | 0.356   | 0.231         | 0.220   | 0.227   | 0.731    | 0.237         | 0.229   | 0.238    | 0.571    | 0.243         | 0.238    | 0.248   | 0.587    |
| soc-youtube       | 495K     | 1M       | 0.418        | 0.795   | 0.404   | 1.678   | 1.082         | 3.806   | 1.069   | 33.877   | 7.795         | 30.482  | 8.435    | 781.582  | 102.463       | 258.912  | 114.663 | -        |
| sc-msdoor         | 415K     | 9M       | 10.726       | 11.456  | 11.095  | 22.242  | 81.904        | 66.695  | 103.154 | 988.826  | 199.684       | 198.704 | 985.887  | 2179.247 | 2360.887      | 2094.814 | -       | -        |
| ca-MathSciNet     | 332K     | 820K     | 0.044        | 0.047   | 0.042   | 0.046   | 0.045         | 0.058   | 0.064   | 0.058    | 0.051         | 0.052   | 0.063    | 0.060    | 0.098         | 0.118    | 0.107   | 0.143    |
| ca-dblp-2012      | 317K     | 1M       | 0.061        | 0.051   | 0.051   | 0.059   | 0.055         | 0.050   | 0.054   | 0.070    | 0.071         | 0.058   | 0.056    | 0.079    | 0.062         | 0.055    | 0.055   | 0.086    |
| ca-citeseer       | 227K     | 814K     | 0.034        | 0.027   | 0.032   | 0.034   | 0.030         | 0.032   | 0.037   | 0.043    | 0.025         | 0.027   | 0.036    | 0.033    | 0.029         | 0.036    | 0.040   | 0.036    |
| ca-dblp-2010      | 226K     | 716K     | 0.023        | 0.022   | 0.025   | 0.028   | 0.022         | 0.022   | 0.027   | 0.026    | 0.023         | 0.025   | 0.028    | 0.028    | 0.025         | 0.024    | 0.032   | 0.024    |
| sc-pwtk           | 217K     | 5M       | 1.679        | 1.684   | 1.593   | 4.598   | 5.801         | 4.756   | 7.262   | 14.455   | 32.956        | 56.107  | 37.546   | 1462.455 | 133.684       | 176.029  | 119.104 | 1619.158 |
| soc-gowalla       | 196K     | 950K     | 0.182        | 0.231   | 0.178   | 0.304   | 0.305         | 0.312   | 0.273   | 1.709    | 0.748         | 1.821   | 0.755    | 71.450   | 6.406         | 19.199   | 8.889   | 2162.518 |
| tech-RL-caida     | 190K     | 607K     | 0.140        | 0.132   | 0.109   | 0.168   | 0.090         | 0.103   | 0.087   | 0.199    | 0.122         | 0.199   | 0.150    | 1.866    | 0.294         | 0.499    | 0.294   | 15.755   |
| sc-shipsec        | 179K     | 2M       | 0.176        | 0.177   | 0.174   | 0.186   | 0.258         | 0.245   | 0.257   | 0.275    | 0.884         | 1.134   | 0.886    | 6.599    | 2.344         | 2.583    | 2.514   | 14.227   |
| web-arabic-2005   | 163K     | 1M       | 0.020        | 0.019   | 0.024   | 0.021   | 0.020         | 0.026   | 0.019   | 0.020    | 0.026         | 0.025   | 0.025    | 0.025    | 0.026         | 0.031    | 0.030   | 0.040    |
| soc-douban        | 154K     | 327K     | 0.030        | 0.031   | 0.032   | 0.043   | 0.029         | 0.030   | 0.038   | 0.045    | 7.509         | 7.214   | 7.268    | 27.132   | 51.643        | 49.694   | 50.022  | -        |
| sc-shipsecl       | 140K     | 1M       | 0.140        | 0.110   | 0.134   | 0.070   | 0.180         | 0.180   | 0.222   | 0.262    | 0.691         | 0.533   | 0.682    | 0.628    | 1.622         | 1.325    | 1.293   | 6.719    |
| web-uk-2005       | 129K     | 11M      | 0.252        | 0.246   | 0.255   | 0.679   | 0.304         | 0.296   | 0.306   | 0.872    | 0.316         | 0.310   | 0.320    | 0.888    | 0.331         | 0.322    | 0.332   | 0.876    |
| web-sk-2005       | 121K     | 334K     | 0.010        | 0.010   | 0.013   | 0.014   | 0.012         | 0.012   | 0.011   | 0.016    | 0.010         | 0.010   | 0.014    | 0.016    | 0.016         | 0.011    | 0.012   | 0.014    |

instances such as `sc-msdoor` and `sc-ldoor`, `-FlowUB` often outperforms `-Branch`, suggesting that the improved branching strategy is particularly effective in reducing the search complexity. This confirms that our tailored branching strategy effectively reduces the search effort and facilitates faster convergence. We also note that the efficiency of BBRes is closely related to the value of  $\lambda_k$ . As shown in Table 2, the running time of BBRes generally increases with  $k$ , which is consistent with the complexity  $O^*(\lambda_k^n)$  and the fact that  $\lambda_k$  increases as  $k$  grows. At the same time, the theoretical gap  $2 - \lambda_k$  (or  $\gamma_k - \lambda_k$ ) becomes smaller as  $k$  increases, indicating that the advantage over the naive  $O^*(2^n)$  bound and the previous  $O^*(\gamma_k^n)$  bound gradually narrows. Nevertheless, even when  $\lambda_k$  is very close to 2 (e.g., at  $k = 20$ ), the improvement can still be meaningful on very large graphs. For example, for a graph with  $10^7$  vertices, we have  $2^{10^7}/\lambda_{20}^{10^7} \approx 54$  and  $\gamma_{20}^{10^7}/\lambda_{20}^{10^7} \approx 2$ . In summary, BBRes consistently achieves the best overall performance by combining tight bounding with effective branching, suggesting that the two techniques are complementary in practice even though their relative contributions vary across instances.

### 6.3 Tests on Dense and Synthetic Graphs

While the previous sections focused on sparse real-world graphs, the maximum  $k$ -defective clique problem is theoretically most challenging on *dense* graphs. To stress-test BBRes under these extreme conditions, we extended our evaluation to the DIMACS benchmark (notoriously hard dense instances) and LFR synthetic graphs [31] (controlled density analysis). Due to space constraints, the detailed DIMACS results and the complete density sensitivity analysis on LFR graphs are deferred to Appendix C of the technical report [50].

The results on DIMACS graphs show that BBRes remains highly competitive on dense benchmark instances and continues to outperform the baselines on most tested values of  $k$ . On the LFR graphs, we examine how graph density affects the relative behavior of `-Branch` and `-FlowUB`. We observe that as the graph density increases, the runtime gap between these two variants gradually narrows. This suggests that on denser graphs, the proposed branching strategy becomes increasingly relevant, and its contribution becomes more comparable to that of the flow-based upper bound. At the same time, BBRes, which combines both components, consistently achieves the best overall performance, with a maximum speedup of up to  $174\times$  over the ablation variants. These results indicate that neither component alone is sufficient to fully capture the benefit on dense graphs; instead, their combination is important for maintaining strong performance in such challenging settings.

## 7 RELATED WORK

The concept of the  $k$ -defective clique was originally formalized by [51]. Early exact methods [26, 27, 43] were practical only on small graphs. The first algorithm capable of handling large graphs was MADEC<sup>+</sup> [14], followed by KDBB [23] and KD-Club [29] which improved empirical performance via preprocessing and upper-bound-based pruning. Recently, kDC [11], MDC [17], kDC2 [12], DnBK [34], and WDC [28] introduced new branching strategies, reduction rules, and heuristics for initial solutions during preprocessing, with the latter four included in our experiments. From a theoretical perspective, MADEC<sup>+</sup> and kDC run in  $O^*(\beta_{2k}^n)$  and  $O^*(\beta_k^n)$  time, respectively,

where  $\beta_i$  is the largest real root of  $x^{i+3} - 2x^{i+2} + 1 = 0$ . kDC2 [12] achieves  $O^*(\beta_{k-1}^n)$ , and MDC [17] achieves  $O^*(\gamma_k^n)$ , where  $\gamma_i$  is the largest real root of  $x^{i+3} - 2x^{i+2} + x^2 - x + 1 = 0$ ; note that  $\gamma_k < \beta_{k-1}$  for  $k \geq 2$ . Distinct from these BB frameworks, DnBK reduces the problem to  $O(n(\Delta\delta)^{2k})$  instances of maximum clique and runs in  $O(n(\Delta\delta)^{2k}\alpha_c^n)$  time, where  $\alpha_c$  is the branching factor of the maximum clique solver ( $\alpha_c = 1.2$  in theory [49],  $\alpha_c \approx 1.33$  in practical implementations [10]<sup>7</sup>); WDC [28] decomposes branching into maximal clique enumeration and  $k$ -defective constraint verification, yielding  $O(n(\Delta\delta)^k\alpha_e^\delta)$  time with  $\alpha_e = 3^{1/3} \approx 1.44$ . Although DnBK and WDC reduce the exponential base, the overhead  $(\Delta\delta)^{\Theta(k)}$  (which is exponential in  $k$ ) limits scalability when  $k$  is not a small constant. In contrast, our BBRes runs in  $O^*(\lambda_k^n)$ , where  $\lambda_k$  is the largest real root of  $x^{k+4} - 2x^{k+3} + x^3 - x + 1 = 0$ , and  $\lambda_k < \gamma_k$  for  $k \geq 1$ , achieving better theoretical complexity for general  $k$  and empirically outperforming kDC2, MDC, DnBK, and WDC (Section 6). In addition, the problem of enumerating all maximal  $k$ -defective cliques has been studied; the Pivot+ algorithm [18] achieves  $O^*(\gamma_k^n)$  time but is less effective for finding the maximum due to limited pruning. The concept has also been extended to bipartite graphs [16, 45].

## 8 CONCLUSION

In this paper, we proposed a new branch-and-bound method BBRes for finding the maximum  $k$ -defective clique. BBRes has the worst-case time complexity of  $O^*(\lambda_k^n)$ , which improves upon the state-of-the-art time complexity of  $O^*(\gamma_k^n)$ , where  $\lambda_k < \gamma_k$ . To boost the practical efficiency, we also proposed a tighter upper bound. Finally, extensive experiments demonstrate that BBRes outperforms state-of-the-art algorithms by several orders of magnitude in practice. In the future, we intend to investigate parallelization strategies for BBRes to handle massive graphs.

## ACKNOWLEDGMENTS

This research is partially supported by the Guangdong Basic and Applied Basic Research Foundation (No. 2025A1515060015), by the Science and Technology Development Fund, Macao SAR (No. 0007/2024/AKP), by the Shenzhen Science and Technology Program (No. GXWD20231129111306002 and KJZD20240903103811016), by the Major Key Project of Pengcheng Laboratory (No. PCL2024A05), and by the Key Laboratory of Interdisciplinary Research of Computation and Economics (Shanghai University of Finance and Economics), Ministry of Education. Kaiqiang Yu is supported by the “111 Center” (No. B26023) and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118).

<sup>7</sup>We derive  $\alpha_c \approx 1.33$  for MC-BRB [10] from the characteristic equation  $x^5 - x^4 - 1 = 0$ , since MC-BRB only branches on vertices with at least four non-neighbors in the candidate set.

## REFERENCES

- [1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. 2016. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems* 55 (2016), 278–288.
- [2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.
- [3] Ravindra K Ahuja and James B Orlin. 1995. A capacity scaling algorithm for the constrained maximum flow problem. *Networks* 25, 2 (1995), 89–98.
- [4] Vladimir Batagelj and Matjaž Zveršnik. 2003. An  $O(m)$  algorithm for cores decomposition of networks. arXiv:cs/0310049 [cs.DS] Accessed: 2026-05-17.
- [5] Punam Bedi and Chhavi Sharma. 2016. Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6, 3 (2016), 115–135.
- [6] Nina Berry, Teresa Ko, Tim Moy, Julianne Smrcka, Jessica Turnley, and Ben Wu. 2004. Emergent clique formation in terrorist recruitment. In *Proceedings of the AAAI Workshop on Agent Organizations: Theory and Practice*. 1198–1208.
- [7] Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. 2002. An exact algorithm for the maximum  $k$ -club problem in an undirected graph. *European Journal of Operational Research* 138, 1 (2002), 21–28.
- [8] Daniel Brélaz. 1979. New methods to color the vertices of a graph. *Commun. ACM* 22, 4 (1979), 251–256.
- [9] Randy Carraghan and Panos M Pardalos. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters* 9, 6 (1990), 375–382.
- [10] Lijun Chang. 2020. Efficient maximum clique computation an enumeration over large sparse graphs. *The VLDB Journal* 29, 5 (2020), 999–1022.
- [11] Lijun Chang. 2023. Efficient maximum  $k$ -defective clique computation with improved time complexity. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 3 (2023), 1–26.
- [12] Lijun Chang. 2024. Maximum defective clique computation: Improved time complexities and practical performance. *Proceedings of the VLDB Endowment* 18, 2 (2024), 200–212.
- [13] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient maximum  $k$ -plex computation over large sparse graphs. *Proceedings of the VLDB Endowment* 16, 2 (2022), 127–139.
- [14] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. 2021. Computing maximum  $k$ -defective cliques in massive graphs. *Computers & Operations Research* 127 (2021), 105131.
- [15] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. 2011. Finding maximal cliques in massive networks. *ACM Transactions on Database Systems (TODS)* 36, 4 (2011), 1–34.
- [16] Donghang Cui, Ronghua Li, Qiangqiang Dai, Hongchao Qin, and Guoren Wang. 2025. On the efficient discovery of maximum  $k$ -defective biclique. arXiv:2506.16121 Accessed: 2026-05-17.
- [17] Qiangqiang Dai, Ronghua Li, Donghang Cui, and Guoren Wang. 2024. Theoretically and practically efficient maximum defective clique search. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 4 (2024), 1–27.
- [18] Qiangqiang Dai, Rong-Hua Li, Meihao Liao, and Guoren Wang. 2023. Maximal defective clique enumeration. *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)* 1, 1 (2023), 1–26.
- [19] Efim A Dinic. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady* 11 (1970), 1277–1280.
- [20] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)* 18 (2013), 3–1.
- [21] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.
- [22] Fedor V Fomin and Dieter Kratsch. 2010. *Exact Exponential Algorithms*. Springer Science & Business Media.
- [23] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. 2022. An exact algorithm with new upper bounds for the maximum  $k$ -defective clique problem in massive sparse graphs. In *Proceedings of AAAI* 10174–10183.
- [24] Shuhao Gao, Kaiqiang Yu, Shengxin Liu, and Cheng Long. 2024. Maximum  $k$ -plex search: An alternated reduction-and-bound method. *Proceedings of the VLDB Endowment* 18, 2 (2024), 363–376.
- [25] Andrew Goldberg and Robert Tarjan. 1987. Solving minimum-cost flow problems by successive approximation. In *Proceedings of STOC*. 7–18.
- [26] Timo Gschwind, Stefan Irnich, Fabio Furini, and Roberto Wolfier Calvo. 2021. A branch-and-price framework for decomposing graphs into relaxed cliques. *INFORMS Journal on Computing* 33, 3 (2021), 1070–1090.
- [27] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. 2018. Maximum weight relaxed cliques and Russian Doll Search revisited. *Discrete Applied Mathematics* 234 (2018), 131–138.
- [28] Jihoon Jang, Yehyun Nam, Kunsoo Park, and Hyunjoon Kim. 2025. Efficient defective clique enumeration and search with worst-case optimal search space. *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)* 3, 6 (2025), 1–28.
- [29] Mingming Jin, Jiongzi Zheng, and Kun He. 2024. KD-Club: An efficient exact algorithm with new coloring-based upper bound for the maximum  $k$ -defective clique problem. In *Proceedings of AAAI*. 20735–20742.
- [30] Jalal Khalil, Da Yan, Guimu Guo, and Lyuheng Yuan. 2022. Parallel mining of large maximal quasi-cliques. *The VLDB Journal* 31, 4 (2022), 649–674.
- [31] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 78, 4 (2008), 046110.
- [32] Kingsly Leung and Christopher Leckie. 2005. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Australasian Conference on Computer Science*. 333–342.
- [33] John M. Lewis and Mihalis Yannakakis. 1980. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. System Sci.* 20, 2 (1980), 219–230.
- [34] Chunyu Luo, Yi Zhou, Zhengren Wang, and Mingyu Xiao. 2024. A faster branching algorithm for the maximum  $k$ -defective clique problem. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 4132–4139.
- [35] Panos M Pardalos and Jue Xue. 1994. The maximum clique problem. *Journal of global Optimization* 4, 3 (1994), 301–328.
- [36] Bharath Pattabiraman, Md Mostofa Ali Patwary, Assefaw H Gebremedhin, Weikeng Liao, and Alok Choudhary. 2015. Fast algorithms for the maximum clique problem on massive graphs with applications to overlapping community detection. *Internet Mathematics* 11, 4-5 (2015), 421–448.
- [37] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2013. On clique relaxation models in network analysis. *European Journal of Operational Research* 226, 1 (2013), 9–18.
- [38] Jian Pei, Daxin Jiang, and Aidong Zhang. 2005. On mining cross-graph quasi-cliques. In *Proceedings of KDD*. 228–238.
- [39] Pablo San Segundo, Alvaro Lopez, and Panos M Pardalos. 2016. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research* 66 (2016), 81–94.
- [40] Stephen B. Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology* 6, 1 (1978), 139–154.
- [41] Etsuji Tomita. 2017. Efficient algorithms for finding maximum and maximal cliques and their applications. In *International workshop on algorithms and computation*. Springer, 3–15.
- [42] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. 2010. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Algorithms and Computation*. Springer, 191–203.
- [43] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. 2013. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications* 56, 1 (2013), 113–130.
- [44] Jianxin Wang, Min Li, Youping Deng, and Yi Pan. 2010. Recent advances in clustering methods for protein interaction networks. *BMC Genomics* 11, Suppl 3 (2010), S10.
- [45] Zhiyi Wang, Lijun Chang, and Jeffrey Xu Yu. 2025. Identifying maximum defective bicliques in large bipartite graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 3710–3723.
- [46] Zhengren Wang, Yi Zhou, Chunyu Luo, and Mingyu Xiao. 2023. A fast maximum  $k$ -plex algorithm parameterized by the degeneracy gap. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 5648–5656.
- [47] Hongbo Xia, Kaiqiang Yu, Shengxin Liu, Cheng Long, and Xun Zhou. 2025. Maximum degree-based quasi-clique search via an iterative framework. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (SIGKDD)*. 3285–3296.
- [48] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. 2017. A fast algorithm to compute maximum  $k$ -plexes in social network analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 919–925.
- [49] Mingyu Xiao and Hiroshi Nagamochi. 2017. Exact algorithms for maximum independent set. *Information and Computation* 255 (2017), 126–146.
- [50] Kewu Yang, Kaiqiang Yu, Shengxin Liu, and Zhaoquan Gu. 2026. Revisiting the maximum defective clique problem: Faster branching and a tighter upper bound (Technical Report). <https://github.com/Thaumaturge2020/BBRes/>.
- [51] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.
- [52] Kaiqiang Yu and Cheng Long. 2021. Graph mining meets fake news detection. In *Data Science for Fake News: Surveys and Perspectives*. Springer, 169–189.
- [53] Kaiqiang Yu and Cheng Long. 2023. Fast maximal quasi-clique enumeration: A pruning and branching co-design approach. *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)* 1, 3 (2023), 1–26.
- [54] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. 2006. Coherent closed quasi-clique discovery from large dense graph databases. In *Proc. of KDD (SIGKDD)*. 797–802.
- [55] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. 2021. Improving maximum  $k$ -plex solver via second-order reduction and graph color bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 12453–12460.