Diversified Top-K Clique Search

Long Yuan[§], Lu Qin[‡§], Xuemin Lin[§♮], Lijun Chang[§], and Wenjie Zhang[§]

[§] *The University of New South Wales, Australia*

[‡]*Centre for QCIS, University of Technology, Sydney, Australia* [♮]*East China Normal University, China*

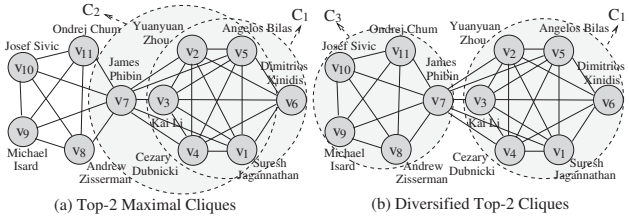
[§]{longyuan,lxue,ljchang,zhangw}@cse.unsw.edu.au; [‡]lu.qin@uts.edu.au

***Abstract*—Maximal clique enumeration is a fundamental problem in graph theory and has been extensively studied. However, maximal clique enumeration is time-consuming in large graphs and always returns enormous cliques with large overlaps. Motivated by this, in this paper, we study the diversified top-k clique search problem which is to find top-k maximal cliques that can cover most number of nodes in the graph. Diversified top-k clique search can be widely used in a lot of applications including community search, motif discovery, and anomaly detection in large graphs. A naive solution for diversified top-k clique search is to keep all maximal cliques in memory and then find k of them that cover most nodes in the graph by using the approximate greedy max k-cover algorithm. However, such a solution is impractical when the graph is large. In this paper, instead of keeping all maximal cliques in memory, we devise an algorithm to maintain k candidates in the process of maximal clique enumeration. Our algorithm has limited memory footprint and can achieve a guaranteed approximation ratio. We also introduce a novel light-weight** PNP-Index**, based on which we design an optimal maximal clique maintenance algorithm. We further explore three optimization strategies to avoid enumerating all maximal cliques and thus largely reduce the computational cost. We conduct extensive performance studies on six real graphs one of which contains** 0.3 **billion edges, and the results demonstrate the high efficiency and effectiveness of our approach.**

I. Introduction

Maximal clique enumeration is a fundamental graph operation. Given an undirected graph G, a *clique* C is a subset of nodes in G in which every two nodes are connected by an edge, and C is a *maximal clique* if no superset of C is a clique. Maximal clique enumeration aims to enumerate all maximal cliques in G. Maximal clique enumeration has been extensively studied in the literature [2, 10–13, 18, 19, 30, 31, 33]. However, maximal clique enumeration is known to be computationally intractable since the number of maximal cliques in a graph G can be exponential to the number of nodes in G [18]. It is difficult to process and analyze such a large number of maximal cliques. A possible solution is to compute top-k maximal cliques ranked by their size, since maximal cliques with larger size are more important and preferred for a user [7]. However, maximal cliques in a graph are usually highly overlapping [33], which significantly reduces the amount of useful information contained in the returned results. Motivated by this, in this paper, we study the problem of *diversified top-k clique search*, which aims to find k maximal cliques that are not only individually large but also lowly overlapping with each other.

Applications. Diversified top-k clique search can be applied in a wide range of applications. For example:



(a) Top-2 Maximal Cliques (b) Diversified Top-2 Cliques

Fig. 1: Part of the Collaboration Network in *DBLP*

(1) Gene expression and motif discovery in molecular biology. In the gene co-expression network, Co-Expression Groups (CEGs) are represented as maximal cliques [36]. Motif discovery in molecular biology requires to obtain the large CEGs with low overlaps, which can be modelled as the diversified top-k clique search problem.

(2) Anomaly detection in complex networks. In this application, cliques are used as signals of rare events and the problem is to find a set of large cliques with low overlaps [8], which can be modelled as the diversified top-k clique search problem.

(3) Community search in social networks. Diversified top-k cliques can serve as the seeds for community search, which can be expanded to lowly overlapping communities [24].

Specifically, given a graph G and an integer k, diversified top-k clique search is to find k maximal cliques that are large and informative in the sense that they together cover most nodes in the graph. We illustrate this by the following example.

Example 1.1: Fig. 1 shows part of the collaboration network in the *DBLP* dataset (http://www.informatik.uni-trier.de/~ley/db/), in which each node represents an author and each edge indicates the co-author relationship between two authors. There are three maximal cliques: $C_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, $C_2 = \{v_1, v_2, v_3, v_4, v_5, v_7\}$, and $C_3 = \{v_7, v_8, v_9, v_{10}, v_{11}\}$. Let $k = 2$. For top-k maximal cliques, we get the result $\mathcal{D}_1 = \{C_1, C_2\}$. For diversified top-$k$ cliques, the result is $\mathcal{D}_2 = \{C_1, C_3\}$. Although $|C_2| > |C_3|$, obviously, $\mathcal{D}_2$ is preferred to $\mathcal{D}_1$, since the two maximal cliques in $\mathcal{D}_1$ are highly overlapping with each other, while the two maximal cliques in $\mathcal{D}_2$ cover all nodes in the graph. □

Challenges. In order to compute diversified top-k cliques, a straightforward solution is to enumerate all maximal cliques first and then apply a greedy max k-cover algorithm [21] to compute an approximate result with a guaranteed approximation ratio. However, such a solution falls short to handle large graphs because clique enumeration is a costly operation and keeping all maximal cliques in memory is infeasible due to the exponential number of maximal cliques in a graph. Therefore, comparing to the conventional k-cover algorithm [21] and its variants (e.g. [25]), the following issues need to be addressed

in order to make diversified top-k clique search practically applicable: (1) How to avoid generating all maximal cliques to compute the final result efficiently? (2) How to avoid keeping all generated maximal cliques in memory? and (3) How to guarantee the result quality when not all maximal cliques are generated and kept in memory?

Contributions. In this paper, we answer all the above questions and make the following contributions.

(1) A simple problem model considering both size and diversity. We formalize the diversified top-k clique search as a problem to maximize the total number of nodes covered by the top-k cliques. The model is simple yet effective since it can be utilized to find large and diversified cliques simultaneously.

(2) An efficient algorithm with bounded memory consumption and a guaranteed approximation ratio. We devise an efficient algorithm based on the maximal clique enumeration algorithm to compute the diversified top-k cliques with a guaranteed approximation ratio. Our algorithm maintains at most k candidate maximal cliques in the memory and keeps updating the candidate set when more promising maximal cliques are generated. The key issue is how to efficiently update the candidate set when new maximal cliques are generated. A basic solution requires $O(|\mathcal{A}| \cdot k \cdot |C_{max}|)$ time, which is costly, to maintain the candidate set, where $\mathcal{A}$ is the set of all maximal cliques, and C_{max} is the maximum clique in the graph. In this paper, we introduce a light-weight online PNP-Index. Based on the PNP-Index, we can reduce such time complexity to $O(\Sigma_{c \in \mathcal{A}}|C|)$, which is optimal in the sense that every generated maximal clique is accessed only once.

(3) Three novel pruning strategies with high pruning power. We explore three novel optimization strategies, namely, global pruning, local pruning, and initial candidate computation, to further improve the efficiency of our algorithm. Global pruning specifies a global search order of nodes during maximal clique computation, such that the algorithm can terminate as soon as expanding a certain node cannot improve the quality of current candidate set. Local pruning adopts a local pruning rule to avoid expanding unpromising partial maximal cliques whenever possible. Initial candidate computation precomputes an initial candidate set using a greedy strategy before enumerating maximal cliques, such that both global pruning and local pruning methods can be applied more effectively. By applying the three optimization strategies, instead of generating all maximal cliques $\mathcal{A}$, our algorithm only needs to generate r maximal cliques, where $r << |\mathcal{A}|$, without sacrificing any result quality, thus the computational cost is largely reduced. Moreover, by using initial candidate computation, the result quality can be improved as well.

(4) Extensive performance studies on six large real datasets. We conduct extensive performance studies using six large real graphs. The experimental results demonstrate that our proposed algorithm can achieve both high effectiveness and high efficiency. Remarkably, in a graph that contains 0.3 billion edges, our proposed algorithm can terminate in one minute, while all existing algorithms fail to output a result.

Outline. Section II provides the formal problem definition and shows the problem hardness. Section III presents two baseline solutions for the problem. Section IV studies our new approach, introduces the novel PNP-Index, and proves its optimality. Section V explores three optimization strategies. Section VI reviews the related work. Section VII evaluates all introduced algorithms using extensive experiments, and Section VIII concludes the paper.

II. Problem Definition

We consider an undirected, unweighted, simple graph $G = (V, E)$, where $V(G)$ represents the set of nodes and $E(G)$ represents the set of edges in G. We denote the number of nodes and number of edges of graph G by n and m respectively, i.e., $n = |V(G)|$ and $m = |E(G)|$. For each node $u \in V(G)$, we use $N(u, G)$ to denote the set of neighbors of u in G, i.e., $N(u, G) = \{v|(u, v) \in E(G)\}$. The degree of a node $u \in V(G)$, denoted by $d(u, G)$, is the number of neighbors of u in G, i.e., $d(u, G) = |N(u, G)|$. For simplicity, we use $N(u)$ and $d(u)$ to denote $N(u, G)$ and $d(u, G)$ respectively if the context is obvious. A subgraph g of G is a graph such that $V(g) \subseteq V(G)$ and $E(g) \subseteq E(G)$. We use $g \subseteq G$ to denote that g is a subgraph of G.

Definition 2.1: (Clique) Given a graph G, a *clique* C in G is a set of nodes such that for any $u \in C$, $v \in C$ ($u \neq v$), we have $(u, v) \in E(G)$. A clique C in G is called a *maximal clique* if there exists no clique C' in G such that $C \subset C'$. □

Definition 2.2: (Coverage $\mathsf{cov}(\mathcal{D})$) Given a set of cliques $\mathcal{D} = \{C_1, C_2, \cdots\}$ in graph G, the *coverage* of $\mathcal{D}$, denoted by $\mathsf{cov}(\mathcal{D})$, is the set of nodes in G covered by the cliques in $\mathcal{D}$, i.e., $\mathsf{cov}(\mathcal{D}) = \bigcup_{C \in \mathcal{D}} C$. □

Problem Statement. Given a graph G and an integer k, the problem of diversified top-k clique search is to compute a set $\mathcal{D}$, such that each $C \in \mathcal{D}$ is a maximal clique, $|\mathcal{D}| \leq k$, and $|\mathsf{cov}(\mathcal{D})|$ is maximized. $\mathcal{D}$ is called diversified top-k cliques.

Problem Hardness. We show the hardness of the problem by considering a simple case: $k = 1$. In this case, the problem becomes the maximum clique problem which is NP-hard [22]. In the literature, the fastest algorithm known to compute the maximum clique runs in time $O(2^{0.249n})$ [28]. Therefore, *the diversified top-k clique search problem is an NP-hard problem*.

III. Baseline Solutions

In the literature, there are several algorithms to enumerate all maximal cliques in a graph G, and an algorithm to enumerate maximal cliques by considering the overlaps among cliques. These algorithms lead to two baseline solutions for diversified top-k clique search. The first solution EnumAll enumerates all maximal cliques in the graph G, and then formulates the problem of diversified top-k clique search as a max k-cover problem which can be solved using a greedy strategy with a bounded approximation ratio. The second solution EnumSub computes a subset $\mathcal{S}$ of maximal cliques in G by considering the overlaps among cliques, and then applies the same greedy strategy as EnumAll to compute the diversified top-k cliques. However, the size of $\mathcal{S}$ cannot be bounded, and the result returned by EnumSub has no approximation guarantee.

A. Algorithm EnumAll

The EnumAll algorithm is shown in Algorithm 1. It first computes the set $\mathcal{A}$ of all maximal cliques using CliqueAll

Algorithm 1 EnumAll(graph $G = (V, E)$, integer k)

```
1: A ← CliqueAll(V, ∅, ∅);
2: return MaxCover(V, A, k);
3: procedure CliqueAll(node set P, node set R, node set X)
4: if P ∪ X = ∅ then
5:     output R as a maximal clique;
6: u ← argmax_{v∈P∪X}{|P ∩ N(v)|};
7: for all v ∈ P \ N(u) do
8:     CliqueAll(P ∩ N(v), R ∪ {v}, X ∩ N(v));
9:     P ← P \ {v}; X ← X ∪ {v};
10: procedure MaxCover(set V, sets S, integer k)
11: D ← ∅; V' ← V;
12: for i = 1 to k do
13:     C ← argmax_{C'∈S−D}{|C' ∩ V'|};
14:     V' ← V' \ C; D ← D ∪ {C};
15: return D;
```

(line 1) and then computes the diversified top-k cliques using the greedy algorithm MaxCover (line 2).

Procedure CliqueAll. CliqueAll is the state-of-the-art algorithm for maximal clique enumeration introduced in [18]. In procedure CliqueAll, it maintains three sets of nodes: P, R and X (line 3). The nodes in R form a partial clique to be expanded. P contains the nodes that are adjacent to all the nodes in R and are potential candidates to be added to the maximal clique. X contains the nodes such that (1) they are adjacent to all the nodes in R, and (2) they have been traversed to form maximal cliques in any previous level of recursion. Adding any node in X to the current partial clique R will result in duplicate cliques. Therefore, nodes in X must be excluded from R. P and X together cover the nodes that are adjacent to all nodes in R. When both P and X become empty (line 4), R cannot be further expanded. Thus, we output R as a maximal clique (line 5). Otherwise, the procedure CliqueAll recursively adds a node from the candidate nodes in P to expand the current partial clique R. Each time a vertex v is added into R, we refine P and X by keeping only the nodes that are also adjacent to v (line 8) and invoke the procedure CliqueAll recursively. P and X are updated after each recursive call (line 9). The pivot node u in line 6 is used to reduce the computational cost of CliqueAll. In principal, every node in P or X can be chosen as a pivot node. Here, we choose the one with the maximum number of neighbors in P because such a pivot node is shown to be computation effective in [18]. As shown in [18], CliqueAll can be implemented in time $O(d \cdot n \cdot 3^{d/3})$ by specifying an order for nodes traversed in line 7, where $d \leq \sqrt{m}$ is the degeneracy of graph G with $d = \max_{g \subseteq G}\{\min_{v \in V(g)}\{d(v, g)\}\}$. It is proved in [18] that CliqueAll is worst-case optimal since there can be $\Theta((n - d) \cdot 3^{d/3})$ maximal cliques in the worst case.

Procedure MaxCover. The MaxCover algorithm (line 10-15) follows the greedy algorithm for the max k-cover problem, which is the problem of selecting k subsets from a collection of subsets such that their union contains as many elements as possible. Given the set of cliques $\mathcal{S}$, the nodes V of G, and an integer k, let $\mathcal{D}$ be the selected cliques, and V' be the set of nodes in V not covered by cliques in $\mathcal{D}$ (line 11). The algorithm greedily selects k cliques into $\mathcal{D}$ (line 12-14). Each time, the clique C to be selected is the one that can cover most nodes in V' (line 13). After selecting C, V' and $\mathcal{D}$ are updated accordingly (line 14). The MaxCover algorithm can achieve an approximation ratio of $(1 - 1/e) \approx 0.632$ which is the best-possible polynomial time approximation algorithm for the k-cover problem as shown in [21].

Algorithm 2 EnumSub(graph $G = (V, E)$, integer k)

```
1: S ← CliqueSub(V, ∅, ∅);
2: return MaxCover(V, S, k);
```

B. Algorithm EnumSub

The EnumSub algorithm is shown in Algorithm 2. Similar to EnumAll, EnumSub first computes a set $\mathcal{S}$ of maximal cliques in G using CliqueSub (line 1) and then invokes the greedy algorithm MaxCover to compute the diversified top-k cliques (line 2). The CliqueSub algorithm is proposed by Wang et al. [33] which computes a subset $\mathcal{S}$ of maximal cliques in a graph G by considering the overlaps among cliques. Suppose $\mathcal{A}$ is the set of all maximal cliques in G, it is guaranteed that for each maximal clique $C \in \mathcal{A}$, there exists a maximal clique $C' \in \mathcal{S}$, such that the similarity between C and C', denoted by $|C \cap C'|/|C|$, is no less than τ, for a parameter τ ($0 < \tau \leq 1$).

The CliqueSub algorithm [33] follows the same framework of the CliqueAll algorithm (see the procedure CliqueAll in Algorithm 1). In [33], the authors observe that the CliqueAll algorithm typically outputs maximal cliques in an order such that two maximal cliques tend to be similar if they are near in the output sequence. Based on such an observation, the CliqueSub algorithm modifies the CliqueAll algorithm, such that each newly computed maximal clique C is reported only if its similarity to the previously reported maximal clique C' is small. In [33], a randomized algorithm and a deterministic algorithm are introduced to solve such a problem and some pruning rules are introduced to prune partial cliques at early stages of the CliqueSub algorithm.

C. Limitations of Baseline Solutions

Comparing EnumAll (Algorithm 1) and EnumSub (Algorithm 2), EnumAll has a bounded approximation ratio, however, it needs to enumerate all maximal cliques; while EnumSub does not need to enumerate all maximal cliques, however, neither the number of reported maximal cliques nor the approximation ratio can be guaranteed. Both EnumAll and EnumSub are not scalable enough to handle large graphs due to the following two reasons:

(R_1) Exponential number of cliques to be kept in memory. Recall that the number of maximal cliques enumerated in EnumAll can achieve $\Theta((n - d) \cdot 3^{d/3})$ for a graph G with degeneracy d, which requires a huge amount of memory to keep all maximal cliques for the greedy algorithm MaxCover to compute the final top-k answers. Although the EnumSub algorithm can reduce the number of reported maximal cliques compared to EnumAll, it still outputs exponential number of maximal cliques without a bound. Therefore, as also verified by our experimental results in Section VII, EnumSub cannot essentially solve the problem when the graph is large.

(R_2) Independent clique enumeration and top-k search processes. The MaxCover algorithm, which is used in both EnumAll and EnumSub, adopts a global selection criteria in each iteration to greedily select the maximal clique that covers most new nodes in the graph as one of the top-k answers. Although MaxCover can achieve a bounded approximation ratio, the global selection criteria requires that all candidate maximal cliques should have been computed before invoking MaxCover. As a result, the maximal clique enumeration procedure (CliqueAll or CliqueSub) and the top-k search procedure

Algorithm 3 EnumKBasic(graph $G = (V, E)$, integer k)

1: $\mathcal{D} \leftarrow \emptyset$;
2: CliqueAll$(V, \emptyset, \emptyset)$ (replace line 5 in Algorithm 1 with CandMaintainBasic(R));
3: **return** $\mathcal{D}$;
4: **procedure** CandMaintainBasic(maximal clique C)
5: **if** $|\mathcal{D}| < k$ **then** { $\mathcal{D} \leftarrow \mathcal{D} \cup \{C\}$; **return**; }
6: $\mathcal{D}' \leftarrow (\mathcal{D} \setminus \{C_{min}(\mathcal{D})\}) \cup \{C\}$;
7: **if** $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$ **then** $\mathcal{D} \leftarrow \mathcal{D}'$;

(MaxCover) have to be invoked independently. However, if we maintain the top-k answers in the process of maximal clique enumeration, there will be more opportunities to prune the unpromising partial cliques at early stages of maximal clique enumeration, and thus largely reduce the computational cost.

IV. A New Approach

In this paper, we devise a new algorithm for diversified top-k clique search, which can overcome the challenges introduced in Section III-C. In the new algorithm, we modify the maximal clique enumeration algorithm CliqueAll (see the procedure CliqueAll in Algorithm 1) to integrate diversified top-k clique search into the process of maximal clique enumeration. Specifically, during the maximal clique enumeration process, we maintain k candidates of the most promising cliques that can maximize the total node coverage, and update the k candidates when new maximal cliques are reported. The new algorithm has the following three advantages:

(A_1) *Low memory consumption*. Unlike EnumAll and EnumSub, our algorithm does not need to keep all enumerated maximal cliques in memory. Instead, our algorithm just needs to keep the graph G and the k candidates of the most promising cliques in the main memory, the size of which is much smaller than the size of all maximal cliques.

(A_2) *Guaranteed result quality*. Our algorithm can achieve a guaranteed approximation ratio of 0.25, and can be much better in practice as verified in the experiments (Section VII).

(A_3) *High pruning power*. By integrating diversified top-k clique search into the process of maximal clique enumeration, we can develop more pruning strategies to avoid expanding unpromising partial cliques at early stages of the maximal clique enumeration algorithm, thus largely improve the efficiency of the algorithm.

In this section, we introduce our basic algorithm and an improved algorithm which target at achieving A_1 and A_2. In the next section, we will focus on the optimization strategies such that A_3 can be achieved.

A. The Basic Algorithm

Before showing our basic algorithm EnumKBasic for diversified top-k clique search, we introduce some definitions.

Definition 4.1: (Private-Node-Set $\mathsf{priv}(C, \mathcal{D})$**)** Given a set of cliques $\mathcal{D} = \{C_1, C_2, \cdots\}$ in graph G, for each clique $C \in \mathcal{D}$, the *private-node-set* of C in $\mathcal{D}$, denoted by $\mathsf{priv}(C, \mathcal{D})$, is the set of nodes in C that are not contained in other cliques in $\mathcal{D}$, i.e., $\mathsf{priv}(C, \mathcal{D}) = C \setminus \mathsf{cov}(\mathcal{D} \setminus \{C\})$. Each $v \in \mathsf{priv}(C, \mathcal{D})$ is called a private node of C in $\mathcal{D}$. □

Definition 4.2: (Min-Cover-Clique $C_{min}(\mathcal{D})$**)** Given a set of cliques $\mathcal{D} = \{C_1, C_2, \cdots\}$ in graph G, the *min-cover-clique* of $\mathcal{D}$, denoted by $C_{min}(\mathcal{D})$, is the clique $C \in \mathcal{D}$ with minimum $|\mathsf{priv}(C, \mathcal{D})|$, i.e., $C_{min}(\mathcal{D}) = \text{argmin}_{C \in \mathcal{D}}\{|\mathsf{priv}(C, \mathcal{D})|\}$. □

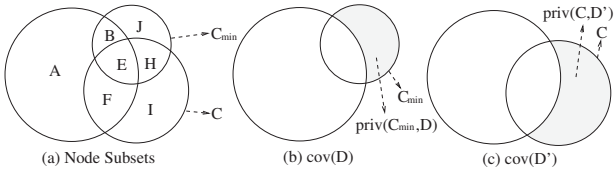


Fig. 2: Illustration for the Proof of Lemma 4.1

Algorithm EnumKBasic. Our basic algorithm EnumKBasic is shown in Algorithm 3. It first initializes the maximal clique set $\mathcal{D}$, which is used to keep the candidates of diversified top-k cliques (line 1). Then it invokes the CliqueAll algorithm to enumerate maximal cliques without keeping all enumerated maximal cliques in memory. Instead, for each enumerated maximal clique C, the procedure CandMaintainBasic is invoked to update $\mathcal{D}$ using C (line 2). Finally, $\mathcal{D}$ is returned as the diversified top-k cliques (line 3).

The procedure CandMaintainBasic is shown in line 4-7 of Algorithm 3. When $|\mathcal{D}| < k$, $\mathcal{D}$ is updated by simply adding C (line 5). Otherwise, we try to replace $C_{min}(\mathcal{D})$ with C to generate a new clique set $\mathcal{D}'$ (line 6). If the number of private nodes of C in the new set $\mathcal{D}'$ is larger than the number of private nodes of $C_{min}(\mathcal{D})$ in $\mathcal{D}$ by $\alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$ for a parameter α $(0 < \alpha \leq 1)$, then $\mathcal{D}$ is updated to be $\mathcal{D}'$ (line 7).

Algorithm Analysis. Obviously, Algorithm 3 only needs to keep graph G and the candidate set $\mathcal{D}$ in the main memory. Next, we analyze the time cost of Algorithm 3. Suppose C_{max} is the maximum clique in G, and $\mathcal{A}$ is the set of all maximal cliques in G, in procedure CandMaintainBasic, we need to compute $C_{min}(\mathcal{D})$, $|\mathsf{priv}(C, \mathcal{D}')|$, $|\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})|$, and $|\mathsf{cov}(\mathcal{D})|$. It is easy to prove that each of the four values can be computed in time $O(k \cdot |C_{max}|)$ by traversing nodes in C and all cliques in $\mathcal{D}$ only once. Suppose $T_{enum}(G)$ is the time to enumerate all maximal cliques in G, the time complexity of Algorithm 3 is $O(T_{enum}(G) + |\mathcal{A}| \cdot k \cdot |C_{max}|)$.

The following lemma shows the quality of the result for the diversified top-k cliques computed using Algorithm 3.

Lemma 4.1: *Given a graph G and an integer k, suppose $\mathcal{D}^*$ is the optimal diversified top-k cliques, and $\mathcal{D}$ is the diversified top-k cliques returned by Algorithm 3 with $\alpha = 1$, we have* $|\mathsf{cov}(\mathcal{D})| \geq 0.25 \times |\mathsf{cov}(\mathcal{D}^*)|$. □

Proof Sketch: The proof is based on a theoretical result derived in [4], which shows the following result:

Given a stream of sets $\mathcal{A} = \{C_1, C_2, \cdots\}$, and an integer k, let $\mathcal{A}_i = \{C_1, C_2, \cdots, C_i\}$ for $i > 0$, and $\mathcal{D}_i = \mathcal{A}_i$ for $0 < i \leq k$. For any $i > k$, we construct $\mathcal{D}_i$ from $\mathcal{D}_{i-1}$ as follows: suppose $\mathcal{D}'_i = (\mathcal{D}_{i-1} \setminus \{C_{min}(\mathcal{D}_{i-1})\}) \cup \{C_i\}$, then $\mathcal{D}_i = \mathcal{D}'_i$ if $|\mathsf{cov}(\mathcal{D}'_i)| > (1 + \frac{1}{k})|\mathsf{cov}(\mathcal{D}_{i-1})|$, and $\mathcal{D}_i = \mathcal{D}_{i-1}$ otherwise. It can be guaranteed that $\mathcal{D}_i$ is a 0.25-approximation solution of the max k-cover problem on $\mathcal{A}_i$.

The above problem has the same setting as our problem. Thus, we only need to prove that when $\alpha = 1$, the condition in line 7 of Algorithm 3, $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})| + \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$, is equivalent to the condition $|\mathsf{cov}(\mathcal{D}')| > (1 + \frac{1}{k})|\mathsf{cov}(\mathcal{D})|$ used in [4], for $\mathcal{D}' = (\mathcal{D} \setminus \{C_{min}(\mathcal{D})\}) \cup \{C\}$. Let $C_{min} = C_{min}(\mathcal{D})$, the relationship of $\mathcal{D}$, $\mathcal{D}'$, C, and C_{min} is illustrated in Fig. 2 using the seven subsets A, B, E, F, J, H, and I. Obviously, the condition $|\mathsf{cov}(\mathcal{D}')| > (1 + \frac{1}{k})|\mathsf{cov}(\mathcal{D})|$

is equivalent to $|A| + |B| + |E| + |F| + |H| + |I| > (1 + \frac{1}{k})(|A| + |B| + |E| + |F| + |H| + |J|)$, which is equivalent to $|H| + |I| > |H| + |J| + \frac{1}{k} \times |\mathsf{cov}(\mathcal{D})|$. Since $|\mathcal{D}| = k$, we have $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min}, \mathcal{D})| + \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. □

Discussion. Note that when $\alpha = 1$, Algorithm 3 is essentially the same as the approach introduced in [4]. However, in this paper, we use a parameter α to allow more flexibility in choosing the size of the newly added cliques to replace the min-cover-clique. In our experiments, we observe that when α is smaller, the algorithm can produce better results without significantly decreasing the efficiency of the algorithm. In addition, in [4], the algorithm needs to compute $|\mathsf{cov}(\mathcal{D}')|$, which is costly, while in Algorithm 3, we only use $|\mathsf{priv}(C, \mathcal{D}')|$ and $|\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})|$, which can lead to an efficient algorithm with the help of an online Private-Node-set Preserved Index (PNP-Index), which will be introduced in the next subsection.

B. Optimal Candidate Maintenance

As discussed in Section IV-A, in addition to the time $O(T_{enum}(G))$ to enumerate all the maximal cliques $\mathcal{A}$, Algorithm 3 needs extra $O(|\mathcal{A}| \cdot k \cdot |C_{max}|)$ time to maintain the candidate set $\mathcal{D}$. It is highly possible that $O(|\mathcal{A}| \cdot k \cdot |C_{max}|) > O(T_{enum}(G))$ when either k or $|C_{max}|$ is large. Therefore, the algorithm is inefficient. The main cost lies on computing the values of $C_{min}(\mathcal{D})$, $|\mathsf{priv}(C, \mathcal{D}')|$, $|\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})|$, and $|\mathsf{cov}(\mathcal{D})|$, each of which needs to traverse all nodes of the cliques in $\mathcal{D}$ in the worst case. In this subsection, we introduce a novel online *Private-Node-set Preserved Index* (PNP-Index), which is used to maintain $|\mathsf{priv}(C, \mathcal{D})|$ for each $C \in \mathcal{D}$, such that $C_{min}(\mathcal{D})$, $|\mathsf{priv}(C, \mathcal{D}')|$, $|\mathsf{priv}(C_{min}(\mathcal{D}), \mathcal{D})|$, and $|\mathsf{cov}(\mathcal{D})|$ can be computed efficiently. We will show that the PNP-Index is compact which only consumes $O(\Sigma_{C \in \mathcal{D}}|C|)$ space, and with PNP-Index, EnumK only takes $O(T_{enum}(G))$ time, which is optimal in the sense that no extra cost is introduced in the time complexity when maintaining $\mathcal{D}$.

Definition 4.3: (Reverse Coverage $\mathsf{rcov}(v, \mathcal{D})$**)** Given a set of cliques $\mathcal{D}$ in graph G, for each node $v \in V(G)$, the *reverse coverage* of v, denoted by $\mathsf{rcov}(v, \mathcal{D})$, is the set of cliques in $\mathcal{D}$ that contain v, i.e., $\mathsf{rcov}(v, \mathcal{D}) = \{C | v \in C, C \in \mathcal{D}\}$. □

Definition 4.4: (Reverse Private-Node-Set $\mathsf{rpriv}(i, \mathcal{D})$**)** Given a set of cliques $\mathcal{D}$ in graph G, for each integer $0 \leq i \leq |V(G)|$, the *reverse private-node-set* of i, denoted by $\mathsf{rpriv}(i, \mathcal{D})$, is the set of cliques C in $\mathcal{D}$ such that $|\mathsf{priv}(C, \mathcal{D})| = i$, i.e., $\mathsf{rpriv}(i, \mathcal{D}) = \{C | C \in \mathcal{D}, |\mathsf{priv}(C, \mathcal{D})| = i\}$. □

The PNP-Index. In the following, for simplicity, we use cov, $\mathsf{priv}(C)$, C_{min}, $\mathsf{rcov}(v)$, and $\mathsf{rpriv}(i)$ to denote $\mathsf{cov}(\mathcal{D})$, $\mathsf{priv}(C, \mathcal{D})$, $C_{min}(\mathcal{D})$, $\mathsf{rcov}(v, \mathcal{D})$, and $\mathsf{rpriv}(i, \mathcal{D})$ respectively, if the context is obvious. The PNP-Index is built on $\mathcal{D}$ and graph G. In addition to $\mathcal{D}$, the PNP-Index consists of five additional components:

- $|\mathsf{priv}(C)|$: the number of private nodes for $C \in \mathcal{D}$.
- $\mathsf{rcov}(v)$: the reverse coverage for each $v \in V(G)$.
- $|\mathsf{cov}|$: the number of nodes covered by $\mathcal{D}$.
- C_{min}: the clique C in $\mathcal{D}$ with minimum $|\mathsf{priv}(C)|$.
- $\mathsf{rpriv}(i)$: reverse private-node-set for $i \leq |V(G)|$.

Where $\mathsf{rcov}(v)$ is used to check whether the node v is privately contained in a certain maximal clique in $\mathcal{D}$, and $\mathsf{rpriv}(i)$ is used to update C_{min} in time $O(|C|)$ (which is independent to k) when processing a newly generated maximal clique C.

Algorithm 4 EnumK(graph $G = (V, E)$, integer k)

```
1: D ← ∅; |cov| ← 0; C_min ← ∅;
2: rcov(v) ← ∅ for all v ∈ V(G); rpriv(i) ← ∅ for all 0 ≤ i ≤ |V(G)|;
3: CliqueAll(V, ∅, ∅) (replace line 5 in Algorithm 1 with CandMaintain(R));
4: return D;
5: procedure CandMaintain(maximal clique C)
6: if |D| < k then { Insert(C); return; }
7: p_new ← |{v ∈ C s.t. |rcov(v)| = 0 or (|rcov(v)| = 1 and v ∈ C_min) }|;
8: if p_new > |priv(C_min)| + α × |cov|/|D| then {Delete(C_min); Insert(C);}
9: procedure Delete(maximal clique C)
10: D ← D \ {C}; remove C from rpriv(|priv(C)|);
11: for all v ∈ C do
12:     rcov(v) ← rcov(v) \ {C};
13:     if |rcov(v)| = 0 then |cov| ← |cov| − 1;
14:     if |rcov(v)| = 1 then
15:         C' ← the maximal clique in rcov(v); |priv(C')| ← |priv(C')| + 1;
16:         move C' from rpriv(|priv(C')| − 1) to rpriv(|priv(C')|);
17: procedure Insert(maximal clique C)
18: D ← D ∪ {C}; |priv(C)| ← 0;
19: for all v ∈ C do
20:     rcov(v) ← rcov(v) ∪ {C};
21:     if |rcov(v)| = 1 then { |priv(C)| ← |priv(C)| + 1; |cov| ← |cov| + 1;}
22:     if |rcov(v)| = 2 then
23:       C' ← the maximal clique in rcov(v) \ {C};|priv(C')| ← |priv(C')| − 1;
24:       move C' from rpriv(|priv(C')| + 1) to rpriv(|priv(C')|);
25: rpriv(|priv(C)|) ← rpriv(|priv(C)|) ∪ {C};
26: for i = 0 to |priv(C)| do
27:     if rpriv(i) ≠ ∅ then {C_min ← an arbitrary clique in rpriv(i); break;}
```

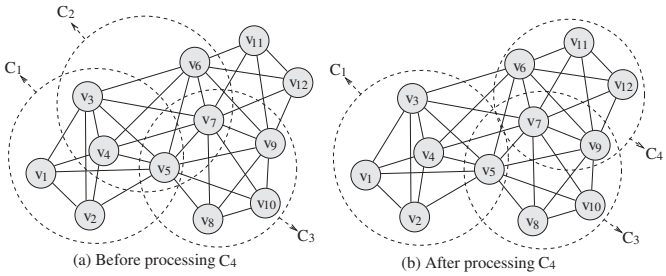
Algorithm EnumK. The new algorithm EnumK is shown in Algorithm 4, which follows the same framework of Algorithm 3 with different candidate maintenance procedures. The new procedure CandMaintain is shown in line 5-8. For each maximal clique C, when $|\mathcal{D}| < k$, it inserts C into $\mathcal{D}$ by invoking a procedure Insert(C) (line 6). Otherwise, it calculates p_{new} which is $|\mathsf{priv}(C, \mathcal{D}')|$ for $\mathcal{D}' = (\mathcal{D} \setminus \{C_{min}\}) \cup \{C\}$ (line 7). If the update condition is satisfied, $\mathcal{D}$ is updated by deleting C_{min} using Delete(C_{min}) and inserting C using Insert(C) (line 8). The key procedures are how to calculate $p_{new} = |\mathsf{priv}(C, \mathcal{D}')|$ (line 7) and how to maintain the PNP-Index using Delete and Insert. Below, we show an example to illustrate the EnumK algorithm and the PNP-Index, and then we introduce the details of the three key procedures.

Example 4.1: Fig. 3 (a) shows a graph G with 12 nodes. Suppose $k = 3$ and the current candidate set is $\mathcal{D} = \{C_1, C_2, C_3\}$, we have $C_{min} = C_2$ with $|\mathsf{priv}(C_{min})| = |\{v_6\}| = 1$, and $|\mathsf{cov}| = 10$. Let $C_4 = \{v_6, v_7, v_9, v_{11}, v_{12}\}$ be the next maximal clique generated, then $\mathcal{D}' = (\mathcal{D} \setminus \{C_2\}) \cup C_4 = \{C_1, C_3, C_4\}$. We have $p_{new} = |\mathsf{priv}(C_4, \mathcal{D}')| = |\{v_6, v_{11}, v_{12}\}| = 3$. Suppose $\alpha = 0.5$, we have $p_{new} = 3 > |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}|}{|\mathcal{D}|} = 2.67$. Therefore, after processing C_4, $\mathcal{D}$ is updated to be $\{C_1, C_3, C_4\}$ as shown in Fig. 3 (b).

Fig. 4 shows the corresponding PNP-Index before and after processing C_4, in which each row corresponds to a maximal clique C and each column corresponds to a node v in G. We also show $|\mathsf{priv}(C)|$ for each maximal clique C in the last column, $|\mathsf{rcov}(v)|$ for each node v in the last row, and $|\mathsf{cov}|$ in the bottom right cell. The columns for private nodes are coloured grey. For each $v \in C$, the corresponding cell is marked $\hat{1}$ if $v \in \mathsf{priv}(C)$, and 1 otherwise. C_{min} is marked a star (*) in the corresponding cell of the last column. □

Computing $\mathsf{priv}(C, \mathcal{D}')$. $\mathsf{priv}(C, \mathcal{D}')$ consists of two parts:

(P_1) The nodes that are contained in C but not contained in $\mathsf{cov}(\mathcal{D})$, e.g., the part denoted by I in Fig. 2. This part can be



(a) Before processing C4 (b) After processing C4

Fig. 3: Candidate Maintenance on Graph G

calculated using $|\{v \in C \text{ s.t. } |\mathsf{rcov}(v)| = 0\}|$.

(P_2) The nodes that are contained in both C and $\mathsf{priv}(C_{min}, \mathcal{D})$, e.g., the part denoted by H in Fig. 2. This part can be calculated using $|\{v \in C \text{ s.t. } |\mathsf{rcov}(v)| = 1 \text{ and } v \in C_{min})\}|$.

In summary, $\mathsf{priv}(C, \mathcal{D}')$ (or p_{new}) can be calculated as $|\{v \in C \text{ s.t. } |\mathsf{rcov}(v)| = 0 \text{ or } (|\mathsf{rcov}(v)| = 1 \text{ and } v \in C_{min}) \}|$, as shown in line 7 of Algorithm 4.

Example 4.2: We show how to compute $p_{new} = \mathsf{priv}(C, \mathcal{D}')$ in Example 4.1 when processing $C = C_4$. The part P_1 can be computed as $|\{v_{11}, v_{12}\}| = 2$, and the part P_2 can be computed as $|\{v_6\}| = 1$, since $v_6 \in C_{min} = C_2$ and $|\mathsf{rcov}(v_6)| = 1$, as shown in Fig. 4. As a result, $\mathsf{priv}(C, \mathcal{D}') = 2 + 1 = 3$. □

Procedure Delete(C). After removing an existing maximal clique C from $\mathcal{D}$, we also need to maintain $\mathsf{rcov}(v)$, $|\mathsf{cov}|$, and $\mathsf{rpriv}(i)$ (for some nodes v and integers i) whose values are changed due to the deletion of C. This is processed by the procedure Delete(C), which is shown in line 9-16 of Algorithm 4.

After removing C from both $\mathcal{D}$ and $\mathsf{rpriv}(|\mathsf{priv}(C)|)$ (line 10), the algorithm visits all nodes $v \in C$ (line 11), and for each such v, it processes the updates in line 12-16. Since C covers node v, $\mathsf{rcov}(v)$ is updated by removing C (line 12). After updating $\mathsf{rcov}(v)$, there are two cases to be considered:

(Case 1) $|\mathsf{rcov}(v)|$ *decreases from* 1 *to* 0: This case indicates that a existing node v which is privately covered by C is not covered by any maximal cliques after removing C, thus, $|\mathsf{cov}|$ decreases by 1 (line 13).

(Case 2) $|\mathsf{rcov}(v)|$ *decreases from* 2 *to* 1: In this case, v becomes a private node for the maximal clique C', where C' is the only maximal clique that contains v after removing C (line 15). Thus $|\mathsf{priv}(C')|$ is updated by increasing 1 (line 15), and C' is removed from $\mathsf{rpriv}(|\mathsf{priv}(C')| - 1)$and added into $\mathsf{rpriv}(|\mathsf{priv}(C')|)$ (line 16).

In other cases when $|\mathsf{rcov}(v)|$ decreases to be larger than 1, no other updates will be triggered.

Example 4.3: Continue Example 4.2. Before adding C_4 into $\mathcal{D}$, we need to remove $C_{min} = C_2$ from $\mathcal{D}$. After removing C_2, v_6 is not covered by any other maximal cliques (the case for $|\mathsf{rcov}(v_6)| = 0$), and thus $|\mathsf{cov}|$ decreases by 1 (line 13); v_3 is only covered by $C_1 \in \mathsf{rcov}(v_3)$ (the case for $|\mathsf{rcov}(v_3)| = 1$), and thus $|\mathsf{priv}(C_1)|$ increases by 1 (line 15). □

Procedure Insert(C). After inserting a new maximal clique C into $\mathcal{D}$, we also need to maintain $\mathsf{priv}(C')$, $\mathsf{rcov}(v)$, $|\mathsf{cov}|$, $\mathsf{rpriv}(i)$, and C_{min} (for some maximal cliques C', nodes v, and integers i) whose values are changed due to the insertion of C. This is processed by the procedure Insert(C), which is shown in line 17-27 of Algorithm 4.

$C \setminus v$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	$\lvert\mathsf{priv}(C)\rvert$
Before processing C_4:													
C_1	1̂	1̂	1	1	1								2
C_2			1	1	1	1̂	1						1^*
C_3					1		1	1̂	1̂	1̂			3
$\lvert\mathsf{rcov}(v)\rvert$	1	1	2	2	3	1	2	1	1	1			$\lvert\mathsf{cov}\rvert$:10
C_4						1	1		1		1	1	
After processing C_4:													
C_1	1̂	1̂	1̂	1̂	1								4
C_3					1		1	1̂	1	1̂			2^*
C_4						1̂	1		1		1̂	1̂	3
$\lvert\mathsf{rcov}(v)\rvert$	1	1	1	1	2	1	2	1	2	1	1	1	$\lvert\mathsf{cov}\rvert$:12

Fig. 4: Example of the PNP-Index

After inserting C into $\mathcal{D}$ and initializing $|\mathsf{priv}(C)|$ to be 0 (line 18), the algorithm visits all nodes $v \in C$ (line 19), and for each such v, it processes the updates in line 20-24. Since C covers node v, $\mathsf{rcov}(v)$ is updated by adding C (line 20). After updating $\mathsf{rcov}(v)$, there are two cases to be considered:

(Case 1) $|\mathsf{rcov}(v)|$ *increases from* 0 *to* 1: This case indicates that a new node v is privately covered by C, thus, both $|\mathsf{priv}(C)|$ and $|\mathsf{cov}|$ increase by 1 (line 21).

(Case 2) $|\mathsf{rcov}(v)|$ *increases from* 1 *to* 2: In this case, v is removed from the private node set of a maximal clique C', where C' is the only maximal clique that contains v before adding C (line 23). Thus $|\mathsf{priv}(C')|$ is updated by decreasing 1 (line 23), and C' is removed from $\mathsf{rpriv}(|\mathsf{priv}(C')| + 1)$ and added into $\mathsf{rpriv}(|\mathsf{priv}(C')|)$ (line 24).

In other cases when $|\mathsf{rcov}(v)|$ increases to be larger than 2, no other updates will be triggered. After traversing all nodes in C, $|\mathsf{priv}(C)|$ is computed. Thus, we need to update $\mathsf{rpriv}(|\mathsf{priv}(C)|)$ by adding a new element C (line 25). Finally, we need to update C_{min} as follows.

Updating C_{min}: A straightforward solution to update C_{min} is to search the maximal clique C' with minimum $\mathsf{priv}(C')$ from $\mathcal{D}$. However, such an operation may introduce an extra cost which is dependent on the value of k, making the algorithm inefficient when k is large. Note that we have maintained the sets $\mathsf{rpriv}(i)$ for all possible i. With all $\mathsf{rpriv}(i)$ $(1 \leq i \leq |V|)$, $|\mathsf{priv}(C_{min})|$ is the first element j such that $\mathsf{rpriv}(j) \neq \emptyset$. We also have $|\mathsf{priv}(C_{min})| \leq |\mathsf{priv}(C)|$. Therefore, we can try every i from 0 to $|\mathsf{priv}(C)|$ (line 26), and find C_{min} in $\mathsf{rpriv}(i)$ for the first i with $\mathsf{rpriv}(i) \neq \emptyset$ (line 27). In this way, C_{min} can be computed in $O(|\mathsf{priv}(C_{min})|) \leq O(|\mathsf{priv}(C)|) \leq O(|C|)$ time which is independent to k.

Example 4.4: Continue Example 4.3. After deleting $C_{min} = C_2$ from $\mathcal{D}$, we insert C_4 into $\mathcal{D}$. After inserting C_4, v_{11} is only covered by C_4 (the case for $|\mathsf{rcov}(v_{11}| = 1$), and thus both $|\mathsf{priv}(C_4)|$ and $|\mathsf{cov}|$ increase by 1 (line 21); v_9 is covered by two maximal cliques $C_3 \in \mathsf{rcov}(v_9)$ and $C_4 \in \mathsf{rcov}(v_9)$ (the case for $|\mathsf{rcov}(v_9| = 2$), and thus $|\mathsf{priv}(C_3)|$ decreases by 1 (line 23) indicating that v_9 is not a private node for C_3. □

Optimality. The following two lemmas show the optimality of Algorithm 4. Lemma 4.2 indicates that the space complexity of the PNP-Index is the same as $\mathcal{D}$. Lemma 4.3 shows that the time complexity of Algorithm 4 is the same as that of maximal clique enumeration (CliqueAll in Algorithm 1). In other words, the maintenance of $\mathcal{D}$ using PNP-Index does not

take extra cost w.r.t. both space and time complexities.

Lemma 4.2: *The* PNP-Index *uses* $O(\Sigma_{C\in\mathcal{D}}|C|)$ *memory.* □

Proof Sketch: Each $C \in \mathcal{D}$ can be stored as a hash set with $O(|C|)$ space s.t. for any $v \in V(G)$, $v \in C$ can be detected in $O(1)$ time. The reverse coverage $\mathsf{rcov}(v)$ for all $v \in V(G)$ consumes $O(\Sigma_{C\in\mathcal{D}}|C|))$ space. The reverse private-node-set $\mathsf{rpriv}(i)$ for all $0 \le i \le |V(G)|$ consumes $O(|\mathcal{D}|)$ space by only keeping those $\mathsf{rpriv}(i) \neq \emptyset$ in memory, and $|\mathsf{priv}(C)|$ for all $C \in \mathcal{D}$ consumes $O(|\mathcal{D}|)$ space. In summary, the total memory used by PNP-Index is $O(\Sigma_{C\in\mathcal{D}}|C|)$. □

Lemma 4.3: *The time complexity of Algorithm 4 is* $O(T_{enum}(G))$*, where* $O(T_{enum}(G))$ *is the time to enumerate all maximal cliques in* G*.* □

Proof Sketch: The main cost of Algorithm 4 is spent on the three key procedures used in CandMaintain to processs maximal clique C, namely, computing $p_{new} = \mathsf{priv}(C, \mathcal{D}')$, $\mathsf{Delete}(C)$, and $\mathsf{Insert}(C)$. Computing p_{new} takes $O(|C|)$ time, and $\mathsf{Delete}(C)$ takes $O(|C|)$ time by traversing every $v \in C$ only once. $\mathsf{Insert}(C)$ also takes $O(|C|)$ time by traversing every $v \in C$ only once in line 19-24, and then computing C_{min} in $O(|C|)$ time in line 26-27. As a result, processing each maximal clique C takes time $O(|C|)$ which is the same as the time of outputting C. Therefore, the time complexity of Algorithm 4 is dominated by $O(T_{enum}(G))$. □

V. Optimization Strategies

A. Solution Overview

Recall that Algorithm 4 computes diversified top-k cliques by processing each maximal clique only once. For each new maximal clique C, it tries to use C to replace the clique with the least number of private nodes in the current candidate set $\mathcal{D}$. Algorithm 4 is efficient to maintain $\mathcal{D}$ using PNP-Index. However, it does not consider the possible opportunities to reduce the set of maximal cliques to be enumerated, and thus reduce the total computational cost. Therefore, in this section, we find three strategies, namely, global pruning, local pruning, and initial candidate computation, to further optimize Algorithm 4 with the aim of reducing the total number of maximal cliques enumerated by Algorithm 4 without reducing the quality of the final answers. Our optimization strategies are based on the following three observations.

(Observation 1) Global pruning: We assign a global priority for all nodes in the graph G when enumerating maximal cliques in Algorithm 4, such that the nodes with high potential to result in large maximal cliques are expanded first. Then the algorithm can terminate early when expanding the remaining nodes does not improve the quality of the current candidates.

(Observation 2) Local pruning: For a partial clique R computed in Algorithm 4, if R has no potential to be expanded to improve the quality of the current candidates, the whole branch expanded from R in Algorithm 4 can be pruned.

(Observation 3) Initial candidate computation: We compute a good initial candidate set $\mathcal{D}$ of k maximal cliques before enumerating all cliques in Algorithm 4. Then both global pruning and local pruning conditions can be satisfied earlier.

Our optimized algorithm EnumKOpt is shown in Algorithm 5, which follows the same framework of Algorithm 4 by adding the three optimization strategies. After initialization (line 1), the algorithm computes the initial candidate set $\mathcal{D}$ using $\mathsf{InitK}(G, k)$ (Section V-D). Then the algorithm traverses the nodes $v \in V$ in non-increasing order of their priorities, denoted by $\mathsf{score}(v)$ (line 4), and stops when the global pruning condition (Section V-B) is satisfied (line 5). Otherwise, the algorithm invokes CliqueK to enumerate maximal cliques expanded from v (line 6). CliqueK (line 9-15) follows the same framework of CliqueAll in Algorithm 1 by adding the local pruning rule (line 11) and replacing line 5 of Algorithm 1 with $\mathsf{CandMaintain}(R)$ which uses the PNP-Index to efficiently maintain the candidate set $\mathcal{D}$ (refer to Algorithm 4). In the following, we will introduce the three optimization strategies.

Algorithm 5 EnumKOpt(graph $G = (V, E)$, integer k)

```
1: line 1-3 of Algorithm 4;
2: D ← InitK(G, k);
3: P ← V; R ← ∅; X ← ∅; u ← argmax_{v∈V}{d(v)};
4: for all v ∈ V \ N(u) in non-increasing order of score(v) do
5:     if |D| = k and GlobalPruning(v) then break;
6:     CliqueK(P ∩ N(v), R ∪ {v}, X ∩ N(v));
7:     P ← P \ {v}; X ← X ∪ {v};
8: return D;
9: procedure CliqueK(P, R, X)
10: if P ∪ X = ∅ then CandMaintain(R);
11: if LocalPruning(P, R) then return;
12: u ← argmax_{v∈P∪X}{|P ∩ N(v)|};
13: for all v ∈ P \ N(u) do
14:     CliqueK(P ∩ N(v), R ∪ {v}, X ∩ N(v));
15:     P ← P \ {v}; X ← X ∪ {v};
```

B. Global Pruning

In global pruning, we need to calculate the priority $\mathsf{score}(v)$ for each node $v \in V(G)$ efficiently, and derive a global pruning condition that makes use of $\mathsf{score}(v)$ such that the algorithm can terminate as early as possible. Before introducing $\mathsf{score}(v)$, we first define **Clique Number** for a given node v or a given set of nodes S.

Definition 5.1: (Clique Number $\omega(v)$ and $\omega(S)$) Given a graph G and a node $v \in V(G)$, the clique number of v in G, denoted by $\omega(v)$, is the size of the maximum clique C in G that contains v, i.e., $\omega(v) = \max_{C\in\mathcal{A}(G),v\in C}\{|C|\}$, where $\mathcal{A}(G)$ is the set of maximal cliques in G. Given a set of nodes $S \subseteq V(G)$, the clique number of S, denoted by $\omega(S)$, is the size of the maximum clique C such that $C \subseteq S$, i.e., $\omega(S) = \max_{C\subseteq S, C \text{ is a clique}}\{|C|\}$. Obviously, $\omega(v) = \omega(N(v))+1$. □

Recall that $\mathsf{score}(v)$ represents the potential size of the maximum clique expanded from v and $\omega(v)$ is exactly the size of the maximum clique that contains v in G. Hence, the best way is to use the clique number $\omega(v)$ as $\mathsf{score}(v)$. However, as shown in Section II, finding the maximum clique in G is an NP-hard problem, and thus computing $\omega(v)$ for all $v \in V(G)$ is also an NP-hard problem. Thus, instead of using $\omega(v)$, we use an upper bound of $\omega(v)$ as $\mathsf{score}(v)$, which is derived from two values, namely, the color number $\mathsf{color}(S)$ for $S \subseteq V(G)$ and the core number $\mathsf{core}(v)$ for $v \in V(G)$.

Definition 5.2: (Color Number $\mathsf{color}(S)$) Given a graph G, a graph coloring $\mathcal{GC}$ is a mapping that maps each node $v \in G$ to a color (a number), such that no two adjacent nodes share the same color. Given a graph coloring $\mathcal{GC}$ for G, and a set of nodes $S \subseteq V(G)$, the color number of S, denoted by $\mathsf{color}(S, \mathcal{GC})$, is the number of distinct colors in S. We use $\mathsf{color}(S)$ to denote $\mathsf{color}(S, \mathcal{GC})$ if the context is obvious. □

Definition 5.3: (Core Number $\mathsf{core}(v)$) Given a graph G and a node v, the core number of v, denoted by $\mathsf{core}(v)$, is the

largest k s.t. there exists a subgraph $g \subseteq G$ with $v \in V(g)$, and for any node $u \in V(g)$, $d(u, g) \geq k$. □

Computing the optimal graph coloring $\mathcal{GC}$ for graph G with minimum $\mathsf{color}(V(G), \mathcal{GC})$ is an NP-hard problem [23]. Thus we adopt the Welsh-Powell algorithm [34], which uses a greedy strategy to compute a graph coloring $\mathcal{GC}$ in $O(m+n)$ time. The core number $\mathsf{core}(v)$ for all nodes $v \in V(G)$ can also be computed in $O(m+n)$ time [6]. For any $v \in V(G)$, we have the following lemma.

Lemma 5.1: $\min\{\mathsf{core}(v), \mathsf{color}(\{v \cup N(v)\})\} \geq \omega(v)$. □

Proof Sketch: The proof is omitted due to lack of space. □

Based on Lemma 5.1, we define $\mathsf{score}(v)$ as follows:

$$\mathsf{score}(v) = \min\{\mathsf{core}(v), \mathsf{color}(\{v \cup N(v)\})\} \quad (1)$$

Given $\mathsf{score}(v)$ for all $v \in G$, the global pruning condition can be defined as follows.

Definition 5.4: (Global Pruning Condition) The global pruning condition $\mathsf{GlobalPruning}(v)$ used in line 5 of Algorithm 5 is defined as: $\mathsf{score}(v) \leq |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. □

Lemma 5.2: *The global pruning condition* $\mathsf{GlobalPruning}(v)$ *defined in Definition 5.4 is correct.* □

Proof Sketch: We only need to prove that once $\mathsf{GlobalPruning}(v)$ is satisfied for a certain v, even if we do not terminate the algorithm (line 5 of Algorithm 5), the candidate set $\mathcal{D}$ will not be updated. We prove this by contradiction. Note that once the condition $\mathsf{score}(v) \leq |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$ is satisfied for a certain v, it will be satisfied for any of the remaining u (the node u that is processed after v in line 4 of Algorithm 5) if $\mathcal{D}$ is not updated, since $\mathsf{score}(u) \leq \mathsf{score}(v)$. Suppose $\mathcal{D}$ is updated to $\mathcal{D}'$ by replacing C_{min} with C when processing u with $\mathsf{score}(u) \leq \mathsf{score}(v)$, by the condition to update $\mathcal{D}$, we have $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. However, by Lemma 5.1, we have $|\mathsf{priv}(C, \mathcal{D}')| \leq \omega(u) \leq \mathsf{score}(u) \leq \mathsf{score}(v)$, and by the global pruning condition, we have $\mathsf{score}(v) \leq |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$, which contradicts with $\mathsf{score}(v) > |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. □

Discussion. Note that the core numbers for all nodes in G and the graph coloring $\mathcal{GC}$ can both be computed in $O(m+n)$ time. Checking the global pruning condition for node v based on Definition 5.4 requires to traverse $N(v)$ only once to compute $\mathsf{color}(\{v \cup N(v)\})$. Therefore, the time complexity of EnumK (Algorithm 4) will not increase after applying global pruning.

C. Local Pruning

In local pruning, we need to define a sufficient condition to stop expanding a partial clique R at early stages of Algorithm 4. We can make use of the information in the two sets P and R in the CliqueAll algorithm, where R is the current partial clique and P is the set of candidate nodes that can be used to expand R to maximal cliques. Intuitively, R can be pruned when $|P|$ is smaller enough, or most nodes in $P \cup R$ have been covered in the current $\mathcal{D}$.

Specifically, given the current candidate set $\mathcal{D}$, C_{min}, P, and R, we define four sets $P_a = P \setminus \mathsf{cov}(\mathcal{D})$, $P_b = P \cap \mathsf{priv}(C_{min})$, $R_a = R \setminus \mathsf{cov}(\mathcal{D})$, and $R_b = R \cap \mathsf{priv}(C_{min})$. The relationship of all the above sets is illustrated in Fig. 5 (a) and (b). The potential of the current partial clique R to be expanded to replace C_{min} depends on the size of the maximum clique in the set $P_a \cup P_b$, i.e., $\omega(P_a \cup P_b)$. However, similar to computing $\omega(v)$ for $v \in V(G)$, computing $\omega(S)$ for $S \subseteq V(S)$ is an NP-hard problem. Thus, instead of using $\omega(S)$ for any $S \subseteq V(S)$, we use an upper bound of $\omega(S)$, denoted by $\mathsf{score}(S)$, which is derived based on the following lemma.

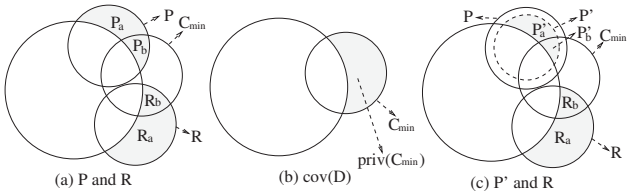


Fig. 5: Illustration for Local Pruning

Lemma 5.3: $\min\{\max_{v \in S} |N(v) \cap S|, \mathsf{color}(S)\} \geq \omega(S)$. □

Proof Sketch: The proof is omitted due to lack of space. □

Based on Lemma 5.3, we define $\mathsf{score}(S)$ as follows:

$$\mathsf{score}(S) = \min\{\max_{v \in S} |N(v) \cap S|, \mathsf{color}(S)\} \quad (2)$$

The local pruning condition is defined as follows.

Definition 5.5: (Local Pruning Condition) The local pruning condition $\mathsf{LocalPruning}(P, R)$ in line 11 of Algorithm 5 is: $\mathsf{score}(P_a \cup P_b) + |R_a \cup R_b| \leq |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. □

Lemma 5.4: *The local pruning condition* $\mathsf{LocalPruning}(P, R)$ *defined in Definition 5.5 is correct.* □

Proof Sketch: We prove that once $\mathsf{LocalPruning}(P, R)$ is satisfied, if we continue expanding R, the candidate set $\mathcal{D}$ will not be updated. We prove this by contradiction. Suppose $\mathcal{D}$ is updated to $\mathcal{D}'$ by replacing C_{min} with $C = R \cup P'$ when expanding R for $P' \subseteq P$, and let $P'_a = P' \setminus \mathsf{cov}(\mathcal{D})$ and $P'_b = P' \cap \mathsf{priv}(C_{min})$, the relationship of R, P', P'_a and P'_b is illustrated in Fig. 5 (c). By the condition to update $\mathcal{D}$, we have $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. However, by Lemma 5.3 and the local pruning condition, we have $|\mathsf{priv}(C, \mathcal{D}')| = |P'_a \cup P'_b \cup R_a \cup R_b| \leq \omega(P'_a \cup P'_b) + |R_a \cup R_b| \leq \mathsf{score}(P'_a \cup P'_b) + |R_a \cup R_b| \leq |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$, which contradicts with $|\mathsf{priv}(C, \mathcal{D}')| > |\mathsf{priv}(C_{min})| + \alpha \times \frac{|\mathsf{cov}(\mathcal{D})|}{|\mathcal{D}|}$. □

Discussion. In local pruning condition, we need to compute $\max_{v \in P_a \cup P_b} |N(v) \cap (P_a \cup P_b)|$ and $\mathsf{color}(P_a \cup P_b)$. Note that after checking $\mathsf{LocalPruning}(P, R)$ (line 11 of Algorithm 5), we need to compute $u \leftarrow \mathrm{argmax}_{v \in P \cup X}\{|P \cap N(v)|\}$ (line 12 of Algorithm 5). Obviously, the cost of computing u is no less than the cost of computing $\max_{v \in P_a \cup P_b} |N(v) \cap (P_a \cup P_b)|$ and $\mathsf{color}(P_a \cup P_b)$. Therefore, the time complexity of EnumK (Algorithm 4) will not increase after applying local pruning.

D. Initial Candidate Computation

Recall that a better initial candidate maximal clique set can potentially help both global pruning and local pruning to gain higher pruning power. In this subsection, we introduce a greedy algorithm to compute an initial candidate maximal clique set $\mathcal{D}$. Intuitively, in a good $\mathcal{D}$, the size $|C|$ of each $C \in \mathcal{D}$ should be large, and the size $|C_i \cap C_j|$ for each pair $C_i \in \mathcal{D}$ and $C_j \in \mathcal{D}$ should be small.

Our algorithm InitK to compute the initial $\mathcal{D}$ is shown in Algorithm 6. Generally speaking, $\mathcal{D}$ is computed by generating

Algorithm 6 InitK(graph $G = (V, E)$, integer k)

```
1: S ← ∅; U ← ∅;
2: for all v ∈ V in non-increasing order of score(v) do
3:     if |S| = η × k then break;
4:     if v ∉ U then
5:         C ← CliqueGreedy(N(v), {v});
6:         S ← S ∪ {C};
7:         for all v ∈ C do U ← U ∪ {v} ∪ N(v);
8: return top-k cliques in S with maximum size;
9: procedure CliqueGreedy(P, R)
10: if P = ∅ then return R;
11: u ← argmax_{v∈P}{min{|P ∩ N(v)|, score(v)}};
12: return CliqueGreedy(P ∩ N(u), R ∪ {u});
```

a set $\mathcal{S}$ of $\eta \times k$ ($\eta \geq 1$) maximal cliques such that each $C \in \mathcal{S}$ is large, and for any two maximal cliques $C_i, C_j \in \mathcal{S}$, $C_i \cap C_j = \emptyset$. In other words, we generate a set of non-overlapping maximal cliques and select the largest k of them to be $\mathcal{D}$. In order to do this, we use U to maintain the set of nodes that are covered by cliques in $\mathcal{S}$ as well as their neighbors in G, i.e., $U = \bigcup_{C \in S, v \in C}\{v\} \cup N(v)$. Both $\mathcal{S}$ and U are initialized to be $\emptyset$ (line 1). Recall that in Section V-B, we show that for each node $v \in V(G)$, the potential size of the maximum clique containing v can be computed using $\mathsf{score}(v)$ (Eq. 1). Thus, in order to find large maximal cliques, we traverse $v \in V$ in non-decreasing order of $\mathsf{score}(v)$ (line 2) and stops the traversal whenever $\eta \times k$ maximal cliques are generated in $\mathcal{S}$ (line 3). In order to avoid overlapping, we compute a maximal clique from each v only if $v \notin U$ (line 4). The non-overlapping condition, i.e., the maximal clique generated from v does not overlap with any maximal cliques in $\mathcal{S}$, is guaranteed because by the definition of U and the condition $v \notin U$, we can guarantee that v is not covered by the current $\mathcal{S}$, and none of the neighbors of v is covered by $\mathcal{S}$. For each such a v, we use a greedy algorithm CliqueGreedy to compute a maximal clique C containing v (line 5), add C into $\mathcal{S}$ (line 6), and maintain U by adding C along with all neighbors of nodes $v \in C$ (line 7). Finally, after $\mathcal{S}$ is generated, we return the top-k cliques in $\mathcal{S}$ with the maximum size as the initial candidate maximal clique set (line 8). Next, we introduce how the greedy algorithm CliqueGreedy works to generate a potentially large maximal clique containing v.

Procedure CliqueGreedy(P, R). The algorithm CliqueGreedy (line 9-12 of Algorithm 6) adopts a recursive approach to generate a maximal clique where R is the current partial clique generated and P is the set of candidate nodes that can be added to R to form larger cliques. The algorithm stops when P is empty (line 10). Otherwise, it selects a node u from P that is likely to form the largest clique with nodes in P. Similar to global pruning (Section V-B), the potential size of such maximum clique in P containing node v can be calculated as $\min\{|P \cap N(v)|, \mathsf{score}(v)\}$ which is obviously an upper bound of the size of the maximum clique formed by v and other nodes in P. Therefore, u can be computed by selecting a node v in P that can maximize the potential size $\min\{|P \cap N(v)|, \mathsf{score}(v)\}$ (line 11). After selecting u, we recursively invoke CliqueGreedy by adding the new selected node u into R, and updating P to be the set of nodes in the original P which are adjacent to u (line 12).

Discussion. Compared to CliqueAll used in EnumK (Algorithm 4) which enumerates all maximal cliques by expanding from every node $v \in V(G)$, in InitK (Algorithm 6) used in EnumKOpt (Algorithm 5), for each node $v \in V(G)$, we only generate one maximal clique. Obviously, the time cost of InitK is not larger than that of CliqueAll. Therefore, the time complexity of EnumK (Algorithm 4) will not increase after applying initial candidate computation. In summary, applying the three optimization strategies in EnumKOpt does not increase the time complexity of the EnumK algorithm.

VI. Related Work

We review the related work to diversified top-k clique search problem from three categories, namely, maximal clique enumeration, max k-cover, and diversified top-k search.

Maximal Clique Enumeration. Maximal clique enumeration is a fundamental graph problem and has been extensively studied. Most algorithms for maximal clique enumeration (e.g., [10] and [2]) are based on backtracking search. [31] and [19] further speedup maximal clique enumeration by selecting good pivots to reduce the search path in backtracking. Maximal clique enumeration in a sparse graph is studied in [11]. A near-optimal algorithm for maximal clique enumeration in a sparse graph is given in [18]. Recently, Wang et al. [33] propose an algorithm to enumerate maximal cliques by taking the overlaps among cliques into consideration. Both [18] and [33] are introduced in details in Section III-A and Section III-B respectively. In addition, parallel maximal clique enumeration is studied in [30], and I/O efficient maximal clique enumeration algorithms are proposed in [12] and [13].

Max k-Cover. As shown in Section III-A, the greedy algorithm to compute the max k-cover can achieve an approximation ratio of $(1 - 1/e)$ which cannot be improved by any polynomial time algorithm unless P=NP [21]. Some other works focus on computing max k-cover in a streaming environment [4, 5, 29, 35]. In this paper, as shown in Section IV-A, we generalize the algorithm introduced in [4] which is an improvement of the algorithm introduced in [29]. In [35], an algorithm is designed in a way that a new set is retained if it has the potential to cover some new nodes in the graph, and an existing set that does not cover any new nodes is removed. After processing all sets, the k retained sets with largest size are returned. In [5], the algorithm maintains multiple lists of sets $\mathcal{D}_{\delta_1}$, $\mathcal{D}_{\delta_2}$, $\cdots$ for $1 < \delta_1 < \delta_2 < \cdots$. Suppose $d_i = (\delta_i/2 - |\mathsf{cov}(\mathcal{D}_{\delta_i})|)/(k - |\mathcal{D}_{\delta_i}|)$, for a new set C, it is inserted into $\mathcal{D}_{\delta_i}$ only if C can cover at least d_i new nodes in $\mathcal{D}_{\delta_i}$. After processing all sets, the $\mathcal{D}_{\delta_i}$ that covers most nodes is returned. The approaches in [35] and [5] have better approximation ratio than [4] and [29] theoretically, however, neither of them can lead to efficient pruning strategies for diversified top-k clique search, thus they are not suitable to handle very large graphs as confirmed in our experiments. Max k-cover computation in MapReduce is studied in [14].

Diversified Top-k Search. Diversified top-k search, which aims at computing the top-k answers that are most relevant to a user query by taking diversity into consideration, has been extensively studied. In the literature, many existing solutions focus on answering the diversified top-k query for a specific problem. For example, diversified top-k document retrieval is studied by Agrawal et al. [1] and Angel and Koudas [3]. Lin et al. [25] study the k most representative skyline problem. Demidova et al. [15] study the diversified keyword query interpretation over structured databases. Diversified top-k graph pattern matching is studied by Fan et al. [20]. However, none of the above approaches can be used to efficiently compute

Dataset G	Type	$\|V(G)\|$	$\|E(G)\|$	Avg Degree
Google	Web	875,713	5,105,039	11.66
Skitter	Physical	1,696,415	11,095,298	13.08
Youtube	Social	3,223,589	12,223,774	7.58
Pokec	Social	1,632,803	30,622,564	37.51
Wiki	Reference	2,936,413	104,673,033	71.29
UK-2002	Web	18,520,486	298,113,762	32.19

Fig. 6: Datasets used in Experiments

diversified top-k cliques. A survey for different query result diversification approaches is provided by Drosou and Pitoura [17]. Some other works focus on a general framework for top-k answer diversification. For example, the general framework to answer diversified top-k queries is studied by Qin et al. [27] and Vieira et al. [32]. Top-k result diversification on a dynamic environment is studied by Minack et al. [26] and Borodin et al. [9]. The complexity of query result diversification is analyzed by Deng and Fan [16]. Nevertheless, the diversity of all the above frameworks is based on the pair-wise dissimilarity of query results, which is not applicable to the diversified top-k clique search problem studied in this paper.

VII. Performance Studies

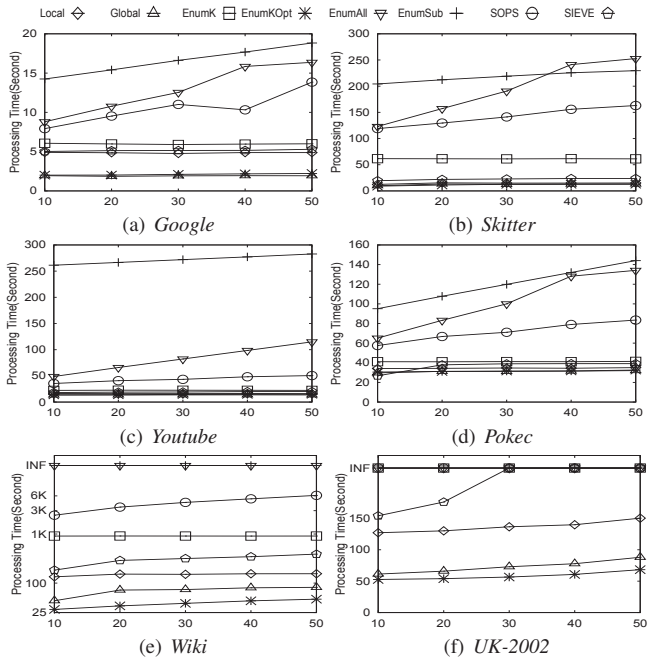
In this section, we show our experimental results. All of our experiments are conducted on a machine with an Intel Xeon 3.4GHz CPU (8 cores) and 32GB main memory running Linux (Red Had Enterprise version 6.4, 64bit).

Datasets. We use six real-world large graphs with different types and graph properties (see Fig. 6) for testing. Among them, *Google*, *Skitter*, and *Pokec* are downloaded from SNAP (http://snap.stanford.edu), *Wiki* and *Youtube* are downloaded from KONECT (http://konect.uni-koblenz.de), and *UK-2002* is downloaded from WEB (http://law.di.unimi.it).

Algorithms. We implement and compare nine algorithms:

- EnumAll: Algorithm 1 (Section III-A).
- EnumSub: Algorithm 2 (Section III-B).
- EnumK: Algorithm 4 (Section IV-B).
- Local: EnumK + local pruning (Section V-C).
- Global: Local + global pruning (Section V-B).
- EnumKOpt: Global + InitK (Section V-D).
- SOPS: Candidate maintenance using method in [29].
- GOPS: Candidate maintenance using method in [35].
- SIEVE: Candidate maintenance using method in [5].

All algorithms are implemented in C++. For EnumSub, we download the source code from the homepage (http://appsrv.cse.cuhk.edu.hk/~jwang/) of the first author of [33]. For SOPS, GOPS, and SIEVE, we apply all optimization techniques introduced in [29], [35], and [5] respectively, and also apply the early termination techniques introduced in Section V whenever possible. For each test, we report the total processing time and the number of nodes covered by the top-k maximal cliques returned. We set the maximum running time for each test to be 10,000 seconds. If a test does not stop in the time limit, or fails due to out of memory exception, we denote the corresponding processing time as INF. For GOPS, we find out that it cannot terminate in the time limit for almost all tests due to its costly



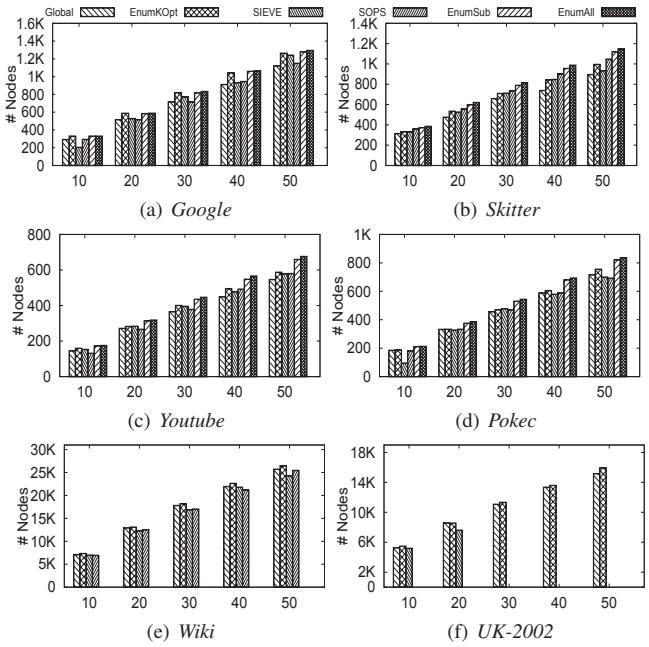
(a) *Google* (b) *Skitter*
(c) *Youtube* (d) *Pokec*
(e) *Wiki* (f) *UK-2002*

Fig. 7: Vary k (Efficiency)

set update operation and low pruning power. Thus, we omit the result for GOPS in the experiments.

Parameters. We vary four parameters in our experiments, namely, k (the top-k value), α (the parameter used in procedure CandMaintain (refer to Algorithm 4)), η (the parameter used in procedure InitK (refer to Algorithm 6)), and $|V|$ (the graph size). k is selected from 10, 20, 30, 40, 50 with a default value of 40. α is selected from $0.1, 0.2, \cdots, 1$ with a default value of 0.3. η is selected from 0, 1, 2, 3, 4, 5 with a default value of 3, where $\eta = 0$ means that no initial candidates are computed. For $|V|$, we generate subgraphs with 20%, 40%, 60%, 80%, and 100% nodes of the original graph for each dataset, with a default value of 100%. Without otherwise specified, when varying a certain parameter, the values of the other parameters are set to their default values.

Exp-1: Vary k (Efficiency). In this experiment, we vary k from 10 to 50. The curves of processing time for all the algorithms in the six datasets are shown in Fig. 7 (a) to Fig. 7 (f) respectively. For all algorithms, when k increases, the processing time increases. EnumAll and EnumSub perform worse than all the other algorithms in all datasets. This is because both EnumAll and EnumSub need to generate a large number of maximal cliques which is costly, and processing the greedy max k-cover algorithm on such a large number of maximal cliques is also costly. SOPS is slower than other algorithms except for EnumAll and EnumSub in all datasets, because the maximal clique swapping operation in SOPS is costly, and early pruning has no large effect on SOPS. Among the other five algorithms, SIEVE is better than EnumK in all datasets, because some early pruning techniques are applied on SIEVE, while EnumK has to enumerate all maximal cliques. However, after applying local pruning, the algorithm Local performs better than SIEVE in all datasets, which shows the high pruning power of the local pruning strategy used in Local. After applying global pruning, our algorithm Global improves Local by 30% to 300% in terms of efficiency. And with initial

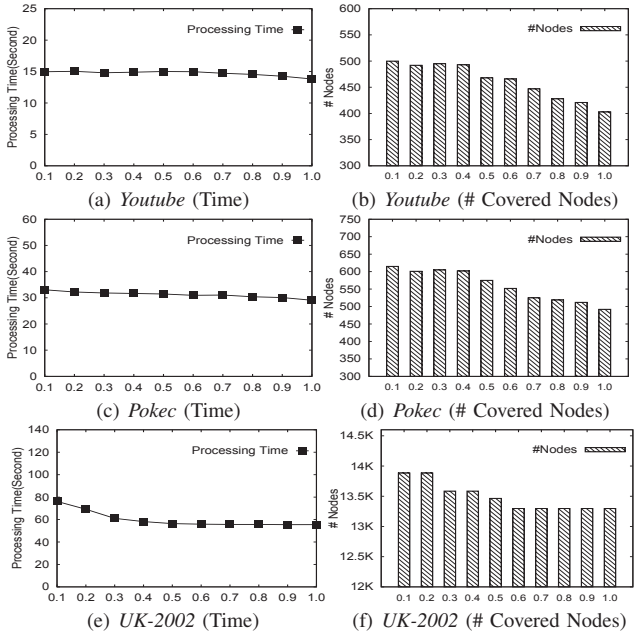


(a) *Google* (b) *Skitter*

(c) *Youtube* (d) *Pokec*

(e) *Wiki* (f) *UK-2002*

Fig. 8: Vary k (Effectiveness)

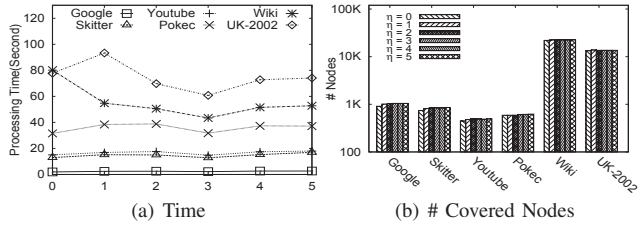


(a) *Youtube* (Time) (b) *Youtube* (# Covered Nodes)

(c) *Pokec* (Time) (d) *Pokec* (# Covered Nodes)

(e) *UK-2002* (Time) (f) *UK-2002* (# Covered Nodes)

Fig. 9: Vary α in Algorithm EnumKOpt



(a) Time (b) # Covered Nodes

Fig. 10: Vary η in Algorithm EnumKOpt

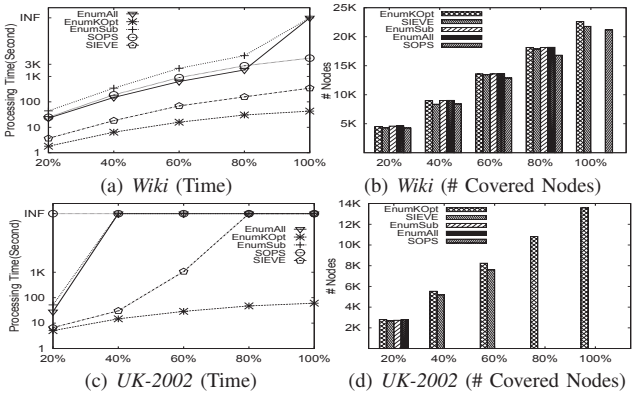
candidate computation, EnumKOpt further improves that of Global. In Fig. 7 (a) to (d), when the size of the dataset is small, the advantage of EnumKOpt is not obvious. However, in Fig. 7 (e) and (f), when the size of the dataset is large, EnumKOpt is much faster than all other algorithms. For example, in *Wiki* (Fig. 7 (e)), EnumKOpt is an order of magnitude faster than SIEVE, and in *UK-2002* (Fig. 7 (f)), when $k > 20$, SIEVE cannot terminate in the time limit, while EnumKOpt can finish in one minute for all k values.

Exp-2: Vary k (Effectiveness). In this experiment, we compare the number of nodes covered by the result returned from different algorithms. The results for the six datasets when varying k from 10 to 50 are shown in Fig. 8 (a) to Fig. 8 (f) respectively. Since the results for EnumK, Local, and Global are the same, we only show the result for Global in Fig. 8. In general, when k increases, the number of covered nodes for all algorithms increases. For the four small datasets *Google*, *Skitter*, *Youtube*, and *Pokec* in Fig. 8 (a) to (d) respectively, EnumAll performs best, followed by EnumSub and EnumKOpt. However, the number of covered nodes in EnumKOpt is very close to that in EnumAll, i.e., no less than 90% of the number of covered nodes in EnumAll in most cases. The other three algorithms Global, SOPS, and SIEVE have similar performance which is worse than EnumKOpt in most cases. Such a result indicates that a good initial candidate set generated in EnumKOpt can improve both efficiency and effectiveness. For the large dataset *Wiki* (Fig. 8 (e)), EnumSub and EnumAll cannot stop in the limited time, and among the other four algorithms, our algorithm EnumKOpt performs best for all k values. For the dataset *UK-2002* (Fig. 8 (f)) only our algorithms Global and EnumKOpt can terminate for all k values, SIEVE can only finish when $k \leq 20$. EnumKOpt performs best in all cases. In the following, for our proposed algorithms EnumK, Local, Global, and EnumKOpt, we only show the results for EnumKOpt, since their relative performances are similar to those shown in Fig. 7 and Fig. 8.

Exp-3: Vary α. We vary α from 0.1 to 1.0 and test both the efficiency and effectiveness of our algorithm EnumKOpt. The results for *Youtube*, *Pokec*, and *UK-2002* are shown in Fig. 9. In general, when α is smaller, the algorithm spends more time, but the corresponding result covers more nodes. For the small datasets *Youtube* (Fig. 9 (a) and (b)) and *Pokec* (Fig. 9 (c) and (d)), the efficiency is not as sensitive to α as the effectiveness. For the large dataset *UK-2002* (Fig. 9 (e) and (f)), both efficiency and effectiveness are sensitive to α when α is small (≤ 0.5), and not sensitive to α when α is large (> 0.5). The results on the other three datasets are similar thus are omitted due to lack of space.

Exp-4: Vary η. We vary η from 0 to 5 and test the procedure InitK in EnumKOpt, where $\eta = 0$ indicates that no InitK is used. The results for efficiency and effectiveness are shown in Fig. 10 (a) and Fig. 10 (b) respectively. In general, when η increases from 0 to 3, the processing time on all datasets tends to decrease, while the number of covered nodes tends to increase. This is because that the pruning power of the global and local pruning strategies increases when η increases from 0 to 3. However, when η further increases, the processing time on all datasets tends to increase, while the number of covered nodes keeps unchanged. The reason is that when η further increases, the procedure InitK takes more time to compute the initial candidate set, while the pruning power of global and local pruning does not increase significantly.

Exp-5: Vary $|V|$. We vary $|V|$ from 20% to 100% of the



(a) *Wiki* (Time) (b) *Wiki* (# Covered Nodes)

(c) *UK-2002* (Time) (d) *UK-2002* (# Covered Nodes)

Fig. 11: Vary $|V|$ (Scalability)

original graph and test the scalability of the algorithms on the two large datasets *Wiki* and *UK-2002*. The results are shown in Fig. 11. In general, when $|V|$ increases, both the processing time and the number of covered nodes increase for all algorithms. EnumKOpt performs best in terms of both efficiency and effectiveness in all tests. Remarkably, in the *Wiki* dataset (Fig. 11 (a) and (b)), the efficiency of EnumKOpt is much better than all other algorithms and the effectiveness of EnumKOpt is even better than that of EnumAll. For *UK-2002* (Fig. 11 (c) and (d)), when $|V|$ increases, the processing time for EnumKOpt increases very stably while that for the other algorithms increases sharply. When $|V| > 60\%$, only EnumKOpt can finish in the time limit. Thus, EnumKOpt has high scalability.

VIII. Conclusion

In this paper, we study the diversified top-k clique search problem, which is to find k maximal cliques that can cover most number of nodes in a graph. We show that it is impractical to keep all maximal cliques in memory before computing the diversified top-k cliques. Therefore, we devise a new algorithm to maintain k candidates during maximal clique enumeration. Our algorithm has limited memory footprint and can achieve a guaranteed approximation ratio. We introduce a novel PNP-Index based on which an optimal candidate maintenance algorithm is designed. We further explore three optimization strategies to avoid enumerating all maximal cliques and thus largely reduce the computational cost. We conduct extensive performance studies on large real graphs to demonstrate the efficiency and effectiveness of our approach.

Acknowledgements. Lu Qin was supported by ARC DE140100999. Xuemin Lin was supported by NSFC61232006, NSFC61021004, ARC DP120104168, and ARC DP140103578. Lijun Chang was supported by ARC DE150100563. Wenjie Zhang was supported by ARC DE120102144 and DP120104168.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proc. of WSDM'09*, 2009.

[2] E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM J. Comput.*, 2(1), 1973.

[3] A. Angel and N. Koudas. Efficient diversity-aware search. In *Proc. of SIGMOD'11*, 2011.

[4] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14), 2012.

[5] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proc. of KDD'14 (to appear)*, 2014.

[6] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[7] H. R. Bernard, P. D. Killworth, and L. Sailer. Informant accuracy in social network data IV: a comparison of clique-level structure in behavioral and cognitive network data. *Social Networks*, 2(3), 1979.

[8] N. Berry, T. Ko, T. Moy, J. Smrcka, J. Turnley, and B. Wu. Emergent Clique Formation in Terrorist Recruitmen. *Workshop on Agent Organizations: Theory and Practice*, 2004.

[9] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proc. of PODS'12*, 2012.

[10] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9), 1973.

[11] L. Chang, J. X. Yu, and L. Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1), 2013.

[12] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.*, 36(4), 2011.

[13] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *Proc. of KDD'12*, 2012.

[14] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *Proc. of WWW'10*, 2010.

[15] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. *DivQ*: diversification for keyword search over structured databases. In *Proc. of SIGIR'10*, 2010.

[16] T. Deng and W. Fan. On the complexity of query result diversification. *ACM Trans. Database Syst.*, 39(2), 2014.

[17] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1), 2010.

[18] D. Eppstein, M. Loffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC (1)*. Springer, 2010.

[19] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *Proc. of SEA'11*, 2011.

[20] W. Fan, X. Wang, and Y. Wu. Diversified top-k graph pattern matching. *PVLDB*, 6(13), 2013.

[21] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4), 1998.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[23] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, 1972.

[24] C. Lee, F. Reid, A. McDaid, and N. Hurley. Detecting highly overlapping community structure by greedy clique expansion. In *Workshop on Social Network Mining and Analysis*, 2010.

[25] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

[26] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *Proc. of SIGIR'11*, 2011.

[27] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11), 2012.

[28] J. Robson. Finding a maximum independent set in time $O(2^{n/4})$, 2001.

[29] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proc. of SDM'09*, 2009.

[30] M. C. Schmidt, N. F. Samatova, K. Thomas, and B.-H. Park. A scalable, parallel algorithm for maximal clique enumeration. *J. Parallel Distrib. Comput.*, 69(4), 2009.

[31] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1), 2006.

[32] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *Proc. of ICDE'11*, 2011.

[33] J. Wang, J. Cheng, and A. W.-C. Fu. Redundancy-aware maximal cliques. In *Proc. of KDD'13*, 2013.

[34] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1), 1967.

[35] H. Yu and D. Yuan. Set coverage problems in a one-pass data stream. In *Proc. of SDM'13*, 2013.

[36] X. Zheng, T. Liu, Z. Yang, and J. Wang. Large cliques in Arabidopsis gene coexpression network and motif discovery. *Journal of Plant Physiology*, 168(6), 2011.