Redundancy-Aware Maximal Cliques

Jia Wang James Cheng Ada Wai-Chee Fu
Department of Computer Science and Engineering
Chinese University of Hong Kong
{jwang, jcheng, adafu}@cse.cuhk.edu.hk

ABSTRACT

Recent research efforts have made notable progress in improving the performance of (exhaustive) maximal clique enumeration (MCE). However, existing algorithms still suffer from exploring the huge search space of MCE. Furthermore, their results are often undesirable as many of the returned maximal cliques have large overlapping parts. This redundancy leads to problems in both computational efficiency and usefulness of MCE.

In this paper, we aim at providing a concise and complete summary of the set of maximal cliques, which is useful to many applications. We propose the notion of τ-*visible* MCE to achieve this goal and design algorithms to realize the notion. Based on the refined output space, we further consider applications including an efficient computation of the top-k results with diversity and an interactive clique exploration process. Our experimental results demonstrate that our approach is capable of producing output of high usability and our algorithms achieve superior efficiency over classic MCE algorithms.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Data mining*; G.2.2 [**DISCRETE MATHEMATICS**]: Graph Theory—*Graph algorithms*

General Terms

Algorithms, Performance

Keywords

Maximal clique enumeration, clique summarization, clique concise representation

1. INTRODUCTION

Let $G = (V, E)$ be a simple undirected graph. A subset of vertices, $C \subseteq V$, is called a *clique* if the subgraph of G induced by C is a complete subgraph, and C is called a *maximal clique* if there exists no clique C' in G such that $C' \supset C$. The process of *Maximal clique enumeration* (**MCE**) is to enumerate the set of all maximal cliques in G.

MCE is a fundamental problem in graph theory and has been extensively studied [5, 6, 21]. Existing works have been focusing on improving the efficiency of MCE in real world graphs and in recent years, a number of new algorithms [8, 9, 10, 13] have been proposed for MCE in large graphs. However, an important issue of MCE has been largely ignored so far, that is, the output size of MCE, i.e., the number of maximal cliques (especially in a large graph), is often exceedingly large so that it has severely thwarted the applications of MCE in practice.

We give a number of problems arisen from the large output size of MCE as follows. First, it is often very expensive to output and store the result set since the set of maximal cliques is too large to be kept in memory and sometimes even on disk. Second, even though the output of MCE may be pipelined to another application program, it is still difficult to process and analyze the large number of maximal cliques. Third, there exists a high level of redundancy in the set of maximal cliques since vertices are often duplicated in multiple maximal cliques.

To address the problem caused by the sheer output size of classic MCE, we propose to compute a *summary* of the set of maximal cliques. In particular, we require a summary to be small in size, able to represent the whole set of maximal cliques with a minimal level of redundancy, and quick to compute (therefore post-processing is not acceptable).

A concise yet complete summary of all maximal cliques in a graph can be useful in a wide range of applications. For example,

- Maximal cliques are widely employed for **anomaly detection** in complex networks [4]. The problem is usually to find a set of large cliques as signals of rare events. To this end, a complete enumeration of large maximal cliques, which can be significantly overlapping with each other, may not be as effective as a summary in which a similar maximal clique is present for each large maximal clique in the whole set.

- Maximal cliques are used to **visualize a large graph** where the cliques are grouped together in the display [15]. For a selected clique, its overlapping cliques are not likely to be chosen. Thus, an exhaustive maximal clique enumeration may again be wasteful, while a smaller set of distinct maximal cliques can be readily applied.

- **Top-*k* computations** are important and useful in information systems, as well in graph databases. One may query the top-k cliques in a graph by a certain quality measure. It is also desirable for the answers to be diverse with minimal redundancy. However, computing such an answer set is often hard,

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

and in fact NP-hard for many quality measures such as maximum coverage. The situation becomes worse when quick response is required or the graph is massive. Fast approximation algorithms, which compute top-k results from the set of maximal cliques with guaranteed quality, are available for certain class of quality measures. However, computing top-k results from a large redundant set is clearly not so efficient as from a good summary.

- In **clique exploration** one wants to flexibly **explore a graph** without fixed objectives. For example, a user may be interested in the groups that involve a particular object. Although an MCE provides the most complete answer, the large number of outputs can easily overwhelm the user. Alternatively, a summary of small size yet containing essential information, from which other information can be (recursively) derived, is often more useful to the user.

We make the following main contributions in this paper.

- We identify the problem of large output size with high redundancy in classic MCE. To address the problem, we introduce the notion of τ-visible MCE, which computes a τ-visible summary that ensures that every maximal clique in the graph is well represented by a maximal clique in the summary.
- We propose a randomized algorithm based on adaptive sampling. The algorithm utilizes important properties of classic backtracking MCE algorithms for efficient redundancy identification and effective search space pruning. We show that the algorithm guarantees to output a summary that well represents the whole set of maximal cliques in expectation, with significantly improved efficiency compared with classic MCE algorithms. We explain intuitively why the method is effective by drawing an analogy to importance sampling.
- We devise a deterministic version of the randomized algorithm, which achieves stronger guarantee of the summary quality. We also introduce an efficient technique that further removes redundant cliques left in the summary to give a more concise τ-visible summary.
- We demonstrate how a τ-visible summary lends solid support to important real world applications, including diversity-aware top-k maximal clique retrieval and a user-friendly clique exploration.

Paper organization. The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 further motivates the problem with an analysis of classic backtracking MCE algorithms and then defines the problem of τ-visible MCE. Section 4 presents the algorithms for τ-visible MCE. Section 5 discusses applications of τ-visible summary. Section 6 reports the experimental results. Section 7 concludes the paper.

2. RELATED WORK

The classical maximal clique enumeration algorithm that is widely in use was proposed in [5] by Bron and Kerbosch. It is a depth-first search algorithm with the property that consecutively generated maximal cliques are likely to be similar. Tomita et al. introduced a variant of the Bron-Kerbosch algorithm which is optimal in the maximum possible number of maximal cliques in [21]. This number was shown to be exponential in [18]. Eppstein et al. [13] introduced a method for listing all maximal cliques in sparse graphs in near-optimal time. For massive graphs, IO-efficient algorithms were studied by Cheng et al. in [10] and [8, 9]. The smallest meaningful clique in a graph is a triangle and the problem of listing all triangles was studied by Chu et al. in [11, 12] for massive graphs.

There are related works on returning meaningful subgraphs that are close to cliques. A quasi-clique Q is almost a clique in the sense that each vertex in Q is connected to at least a portion of γ ($0 < \gamma \leq 1$) of the other vertices in Q. Matsuda et al. [17] proposed an approximation algorithm to find a minimum number of quasi-cliques to cover all vertices in the given graph. For mining all quasi-cliques with given size and γ thresholds, the fastest algorithm is due to Liu et al. [16]. Pei et al. [20] studied the problem of cross-graph quasi-cliques. Abello et al. considered γ-cliques in a graph G, where a γ-clique S is a set of vertices such that the induced subgraph in G is connected and contains at least $\gamma \times \binom{|S|}{2}$ edges, a randomized adaptive search algorithm was proposed to find a γ-clique with the maximum size [1]. Note that the objective of the above methods is to generate maximal cliques as well as subgraphs that are not cliques but close to cliques. Our objective is to alleviate the problem of returning too many maximal cliques to the user in the case where the cliques are highly overlapping and most of the results do not add values for the user. Our returned result is a subset of the set of all maximal cliques.

The problem of result diversification has been well studied for information retrieval. Without system support, a user may need to submit multiple queries to retrieve results of different natures. Vee et al. [22] formally defined diversity and introduced techniques that guarantee diversity. In [3], the problem of DIVERSIFY(K) is to return the top K results that can best cover different possible categorizations of the query. A greedy algorithm is proposed with an approximation guarantee of $(1 - 1/e)$. Another way to select meaningful results is based on typical instances which represent a category of interest [14].

A similar problem of finding diverse results arises in data mining for discovering frequent patterns. In many applications the returned set of frequent patterns from data mining is very large and contains much redundancy. Similar to the problem in information retrieval, the mining result becomes unmanageable for the user. In [2], the aim is to find a small number of item sets which can best approximate a collection of sets. Yan et al. [25] considered how to summarize a large set of patterns using K representatives. Xin et al. [24] considered the mining of redundancy-aware top-k patterns.

Our τ-visible summary may be applied to visualize a large graph [15] using local substructures, visualizing a large graph can also use global structures such as k-trusses [23] and k-cores [7].

3. MOTIVATION AND PROBLEM DEFINITION

We first briefly review the classic recursive backtracking procedure that exhaustively enumerates all maximal cliques in an undirected graph G. We identify the drawbacks of the traditional MCE computation and then propose a new type of *redundancy-aware maximal cliques*.

3.1 Classic MCE and Its Drawbacks

We denote by $\mathcal{N}(v)$ the set of neighbors of a vertex v in G, and by $\mathcal{M}(G)$ the set of all maximal cliques in G. A classic MCE algorithm relies on recursive calls to Procedure $ProcMCE$, which is outlined in Procedure 1. The algorithm takes a graph G as input and initially invokes $ProcMCE(\emptyset, V, \emptyset)$. The recursive procedure finally returns $\mathcal{M}(G)$.

In Procedure 1, the basic idea is to recursively backtrack to add a vertex from the set of candidate vertices in T to grow the current

Procedure 1: ProcMCE(C, T, D)

```
1  if T = ∅ and D = ∅ then
2  |   Output C as a maximal clique;
3  |   return;
4  end
5  Choose a pivot vertex v_p from (T ∪ D);
6  T' ← T \ N(v_p);
7  foreach v ∈ T' do
8  |   T ← T \ {v};
9  |   Call ProcMCE(C ∪ {v}, T ∩ N(v), D ∩ N(v));
10 |   D ← D ∪ {v};
11 end
```

clique C. A vertex v is a candidate to C if and only if v is a neighbor of all vertices in C. Each time when C is augmented by a vertex v, we refine T by keeping only the vertices that are also neighbors of v. When T becomes empty, C cannot be further grown. At this point, we need to check whether C is indeed maximal. To do this, we maintain a set D which keeps the set of vertices that are neighbors of all vertices in C and have been outputted as part of some maximal clique earlier, i.e., the recursive procedure has outputted some maximal clique $C' \supseteq (C \cup \{v\})$ earlier, where $v \in D$. Thus, if D is not empty, C is not a maximal clique; otherwise, we output C as a maximal clique.

The pivot vertex v_p is used for pruning such that we do not need to start growing a maximal clique from any neighbor v of v_p, because a maximal clique containing v can be enumerated from either v_p or some neighbor u of v where u is not a neighbor of v_p.

Drawbacks. The above algorithm achieves the optimal worst-case time complexity [21], which is $O(3^{|V|/3})$. For d-degenerate graphs, the complexity is reduced to $O(|V|3^{d/3})$ [13]. In either case, however, the number of maximal cliques is exponential in either $|V|$ or d, which is not practical for large graphs. For most real world graphs, the number of maximal cliques can be significantly larger than the size of the input graph.

The time taken to compute and output the set of all maximal cliques is generally large in practice. In fact, even if we can tolerate the prolonged running time, the storage space required is often large and the sheer number of maximal cliques also makes it impractical to use them in real applications.

Another problem with the set of maximal cliques is that there are a significant level of redundancy as many maximal cliques share a large portion of common vertices. Thus, it is not very useful to report all these cliques since they deliver a large amount of duplicate information. Such redundancy also slows down the computation significantly without delivering much more new information during the MCE process. Therefore, it is questionable that we always follow the classic exhaustive search approach for maximal clique reporting.

3.2 Redundancy-Aware Maximal Cliques

To address the drawbacks of the classic MCE approach, we propose to compute *redundancy-aware maximal cliques* and we show that the computation process is efficient and the result set is able to convey information possessed by the set of all maximal cliques effectively.

We first introduce the notion of *visibility* that measures how well each maximal clique is revealed by a subset $\mathcal{S}$ of the set $\mathcal{M}(G)$.

DEFINITION 1 (VISIBILITY). *Let $\mathcal{S} \subseteq \mathcal{M}(G)$ be any subset of maximal cliques of G. We define the* visibility *of a maximal clique $C \in \mathcal{M}(G)$ with respect to $\mathcal{S}$, denoted by $\mathcal{V}_{\mathcal{S}}(C)$, to be the maximum ratio of coverage of C by any $C' \in \mathcal{S}$, i.e., $\mathcal{V}_{\mathcal{S}}(C) = \max_{C' \in \mathcal{S}} \frac{|C \cap C'|}{|C|}$.*

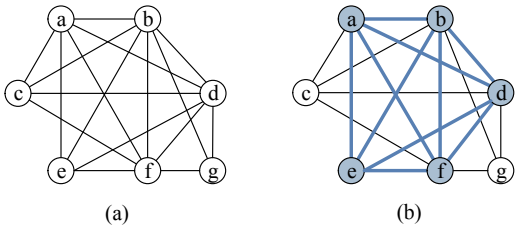

Figure 1: Illustration of visibility and τ-visible summary

When the context is clear regarding $\mathcal{S}$, we write $\mathcal{V}(C)$ instead of $\mathcal{V}_{\mathcal{S}}(C)$ for short. On the basis of visibility, we define the problem of **τ-visible MCE** as follows.

DEFINITION 2 (τ-VISIBLE MCE). *The problem of* τ-visible MCE *is to compute a* τ-visible summary, *$\mathcal{S} \subseteq \mathcal{M}(G)$, such that the visibility of each $C \in \mathcal{M}(G)$ with respect to $\mathcal{S}$ is at least τ, i.e., $\mathcal{V}(C) \geq \tau$.*

The notion of τ-visible MCE provides us with the flexibility of trading the completeness of $\mathcal{M}(G)$ for a τ-visible summary of $\mathcal{M}(G)$ that is more usable and more efficient to compute, in a controllable manner by adjusting the value of τ.

EXAMPLE 1. *In Figure 1(a), we show a graph with 3 maximal cliques, namely C_1 = {a,b,c,d,f}, C_2 = {a,b,d,e,f}, and C_3 = {b,d,f,g}. It can be seen that the three cliques overlap in most of the vertices. If we set $\mathcal{S} = \{C_2\}$ as shown in bold in Figure 1(b), then the visibility of C_1, $\mathcal{V}_{\mathcal{S}}(C_1)$, is 0.8, and that of C_3 is 0.75. Hence $\{C_2\}$ is a 0.75-visible summary. Similarly, $\{C_2, C_3\}$ is a 0.8-visible summary.* □

In the rest of the paper, we will discuss (1) efficient computation of τ-visible MCE, for which we show that our algorithms drastically outperform that for classic MCE, and (2) retrieval of the k best (top-k) maximal cliques based on τ-visible MCE.

4. τ-VISIBLE MCE

To reduce the redundancy in the result set returned by classic MCE, we first propose a randomized algorithm to obtain a sample summary of $\mathcal{M}(G)$ with an expected visibility of at least τ. We then devise a method with similar powerful pruning technique to derandomize the randomized algorithm. The new algorithm computes an exact τ-visible summary.

4.1 A Sampling Approach

A common approach to reducing the output size is to use a sample of smaller size to summarize the whole output space. However, in the case of maximal cliques, a simple uniform sampling can miss important details while the redundancy may not be effectively reduced. To obtain a quality sample, we investigate the output pattern of classic MCE algorithms.

Our randomized algorithm is based on the important observation that although the maximal cliques can be huge in number, an MCE algorithm (based on backtracking) typically outputs in an order such that two cliques tend to be similar if they are near in the output sequence. This is because the searching process follows a depth-first order. This property allows us to (1) avoid reporting a

Algorithm 2: Summarization by Sampling

Input: Graph G, probability function s, visibility threshold τ
Output: A summary $\mathcal{S}$ of $\mathcal{M}(G)$

1 $\mathcal{S} \leftarrow \emptyset$;
2 Let C' be the maximal clique previously included in $\mathcal{S}$;
3 $C' \leftarrow \emptyset$;
4 Call $ProcMCE(\emptyset, V, \emptyset)$, during the process when entering each recursion of $ProcMCE(C, T, D)$, process Lines 5-9:
5 Let $\overline{d}$ be an upper bound of the depth of the search subtree $\mathcal{T}$ to be grown by a complete recursive evaluation of the current $ProcMCE(C, T, D)$ procedure; let P be any maximal clique to be found during the recursive process, and $t = |P \setminus C|$; let $Y = T \setminus C'$; and assume that y_t vertices in Y and $(t - y_t)$ vertices in $T \cap C'$ are used to grow C to P; let $\overline{y_t}$ be an upper bound of y_t;
6 $\overline{l} \leftarrow |C| + \overline{d}$;
7 $\tilde{r} \leftarrow \min_{1 \le t \le \overline{d}} (|C \cap C'| + \max\{t - \overline{y_t}, 0\})/(|C| + t)$;
8 Prune $\mathcal{T}$ with probability 1-$\sqrt[\overline{l}]{s(\tilde{r})}$;
9 Execute $ProcMCE(C, T, D)$ if $\mathcal{T}$ is not pruned, for each C computed to be a maximal clique, do: include C in $\mathcal{S}$ with probability $\sqrt[|C|]{s(r)}$, where $r = |C' \cap C|/|C|$, and in case C is included, $C' \leftarrow C$

clique largely overlapping with the previously reported clique and (2) effectively prune the search space. Both (1) and (2) can be performed with only the local knowledge of the search tree without examining all the previous outputs.

As outlined in Algorithm 2, the processes of sampling and pruning are embedded in a backtracking-based MCE algorithm. Specifically, pruning is performed when the MCE process is starting a new search subtree $\mathcal{T}$ (Lines 5-8), while sampling is executed when a maximal clique is found by the MCE algorithm (on a non-pruned branch of the search tree) (Line 9).

First, we discuss sampling (Line 9). Let us consider whether a found maximal clique C should be reported. Let C' be the maximal clique previously added to $\mathcal{S}$. We retain C with a probability that is adaptive to its similarity to C'. Formally, we add C to $\mathcal{S}$ with probability $\sqrt[|C|]{s(r)}$, where $r = |C \cap C'|/|C|$, and $s : [0, 1) \mapsto [0, 1]$ is the sampling probability function. We require s to be monotonically decreasing. We will show later in Theorem 1 that the probability of each C chosen in $\mathcal{S}$ is at least $s(r)$. In other words, the larger portion of C is covered by its predecessor C', the lower probability we add C to $\mathcal{S}$. We remark that without the local similarity between maximal cliques in the output sequence of classic MCE, we will need a costly computation of $|C \cap C'|$ for all $C' \in \mathcal{S}$.

Next, we discuss how we employ pruning to speedup the search whenever the MCE process is going to start a new search subtree $\mathcal{T}$ (Lines 5-8). Let d be the depth of $\mathcal{T}$, and $\overline{d}$ be an upper bound of d (we will discuss how we compute $\overline{d}$ shortly). Then, the size of each clique P to be generated from $\mathcal{T}$ is at most $\overline{l} = |C| + \overline{d}$.

On the other hand, we want to have $|C' \cap P|$ lower-bounded. Let $Y = T \setminus C'$, and $t = |P \setminus C|$ (i.e., t vertices are used to grow C to P). Part of the t vertices are taken from Y and the rest are from $T \cap C'$. Assume that y_t out of t vertices are taken from Y, and let $\overline{y_t}$ be an upper bound of y_t ($\overline{y_t}$ will be estimated similarly as $\overline{d}$). First, C is known and we know $|C' \cap P| \ge |C' \cap C|$. Second, we know that y_t (out of t) vertices in P (or precisely in $P \setminus C$) are not in $C' \cap P$ since these y_t vertices are taken from $Y = T \setminus C'$. Thus, we have $|C' \cap P| \ge |C \cap C'| + \max\{t - \overline{y_t}, 0\}$.

Finally, by enumerating t, we can lower bound the ratio of P covered by C' by

$$\tilde{r} = \min_{1 \le t \le \overline{d}} \frac{|C \cap C'| + \max\{t - \overline{y_t}, 0\}}{|C| + t}.$$

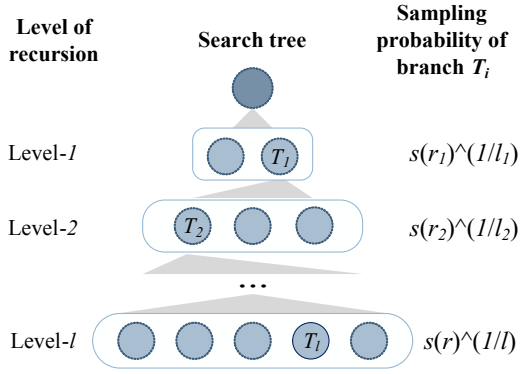

Figure 2: Illustration of summarization by sampling with maximal clique C contained in $\mathcal{T}_l$, where $l = |C|$, and r_i (resp. l_i) is r (resp. l) in Algorithm 2 associated with $\mathcal{T}_i$.

The search space represented by the search subtree $\mathcal{T}$ is only expanded with probability $\sqrt[\overline{l}]{s(\tilde{r})}$. This probability is devised based on the idea as illustrated in Figure 2, which can be viewed as multi-layer sampling over levels of the search tree, where the samples from an upper level are cascaded downwards to the next lower level.

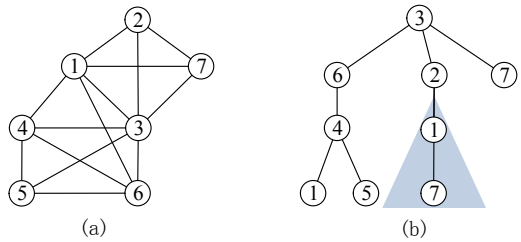

Figure 3: Local similarity in running ProcMCE on a graph: (a) input graph G, (b) search tree for G

EXAMPLE 2. *Consider the execution of $ProcMCE(\phi, V, \phi)$ on the graph in Figure 3(a), where $V = \{1, 2, 3, 4, 5, 6, 7\}$. At Line 5 of $ProcMCE$, we choose a pivot v_p to maximize $V \cap N(v_p)$, where $N(v_p)$ is the set of neighbors of v_p excluding v_p. Hence we choose $v_p = 3$. $T' = \{3\}$. At Line 9, we call $ProcMCE(\{3\}, V \setminus \{3\}, \phi)$. We can visualize this as the top level in Figure 3(b). In the next recursion, let us choose $v_p = 6$. Then we have $T' = \{6, 2, 7\}$. The for loop at Line 7 is shown as the second level in the search tree in Figure 3(b). We first call $ProcMCE(\{3, 6\}, \{5, 1, 4\}, \phi)$. 4 will be chosen as our next pivot, so the next recursive call is $ProcMCE(\{3, 6, 4\}, \{1, 5\}, \phi)$, and further recursively we call $ProcMCE(\{3, 6, 4, 1\}, \phi, \phi)$, at which point a maximal clique is generated. The next clique generated is $\{3, 6, 4, 5\}$. The redundancy with the previously generated clique is 0.75.* □

Before a detailed discussion of how $\overline{d}$ (and similarly $\overline{y_t}$) is estimated, we first present the nice properties of our algorithm in the following theorem.

THEOREM 1. *Let $0 \le \tau \le 1$ be a constant specified by users. By choosing s to be*

$$s(r) = \frac{(1 - r)(2 - \tau)}{2 - r - \tau}$$

Algorithm 2 samples a maximal clique with probability at least $s(r)$, and returns a sample summary $\mathcal{S}$ such that the expected visibility $\mathbb{E}[\mathcal{V}(C)]$ of each $C \in \mathcal{M}(G)$ is at least τ.

PROOF. Consider the event whether or not a maximal clique C is sampled. Since C is sampled only when the search subtree growing to the leaf that represents C is pruned. Let the sequence of nodes from the root to the leaf in the search tree be $v_1, v_2, \ldots, v_k$, where $k = |C|$, and particularly v_k is the leaf node that represents C. Referring to Algorithm 2, let the corresponding sequences of $\overline{l}$ and $\tilde{r}$ defined for the subtrees rooted at $v_1, v_2, \ldots, v_k$ be $\overline{l_1}, \overline{l_2}, \ldots, \overline{l_k}$ and $\tilde{r_1}, \tilde{r_2}, \ldots, \tilde{r_k}$, respectively. Let $r^* = \max\{\tilde{r_1}, \ldots, \tilde{r_l}\}$. As the sampling probability function s is monotonically decreasing, $s(r^*) \leq s(\tilde{r_i})$ for $1 \leq i \leq k$, and $s(r^*) \leq s(r)$. Since $\overline{l}$ is an upper bound of the depth of some search subtree $\mathcal{T}_i$ rooted at v_i, we have $0 < k \leq \overline{l_i}$. It follows that

$$\begin{aligned}
\Pr[C \in \mathcal{S}] &= \prod_{1 \leq i \leq k} \Pr[\mathcal{T}_i \text{ not pruned}] \\
&= \left(\prod_{1 \leq i < k} \sqrt[\overline{l_i}]{s(\tilde{r_i})} \right) \cdot \sqrt[k]{s(r)} \\
&\geq \left(\prod_{1 \leq i < k} \sqrt[k]{s(r^*)} \right) \cdot \sqrt[k]{s(r^*)} \\
&= s(r^*)
\end{aligned}$$

If C is discarded, then there exists $C' \in \mathcal{S}$ such that $|C \cap C'|/|C| \geq r^*$. Then

$$\begin{aligned}
\mathbb{E}[\mathcal{V}(C)] &\geq 1 \cdot \Pr[C \in \mathcal{S}] + r^* \Pr[C \notin \mathcal{S}] \\
&\geq s(r^*) + r^*(1 - s(r^*))
\end{aligned}$$

In case that $r^* \geq \tau$,

$$\mathbb{E}[\mathcal{V}(C)] \geq \tau \cdot s(r^*) + \tau(1 - s(r^*)) = \tau$$

Otherwise, $0 \leq r^* < \tau$,

$$s(r^*) = \frac{(1 - r^*)(2 - \tau)}{2 - r^* - \tau} > \frac{\tau - r^* + (1 - \tau)}{1 - r^* + (1 - \tau)} > \frac{\tau - r^*}{1 - r^*}$$

and

$$\mathbb{E}[\mathcal{V}(C)] > 1 \cdot \frac{\tau - r^*}{1 - r^*} + r^* \cdot (1 - \frac{\tau - r^*}{1 - r^*}) = \tau$$

□

As shown by Theorem 1, the sampling algorithm is capable of not only reducing the output size, but also providing a summary $\mathcal{S}$ with guaranteed quality by selecting a proper sampling probability function s. Note that in Theorem 1, $r < 1$ because by definition a maximal clique C cannot be contained in any other clique. The function s chosen in Theorem 1 is monotonically decreasing and has the property that

$$s(r) = \begin{cases} 1 & \text{when } r = 0 \\ 0 & \text{when } r = 1 \end{cases}$$

as illustrated in Figure 4. It implies that a maximal clique is almost always discarded when it is nearly identical to a maximal clique previously added to the summary $\mathcal{S}$.

Now we provide the details of computing $\overline{d}$ and $\overline{y_t}$. Recall d is the depth of the search subtree $\mathcal{T}$ to be grown by a complete recursive evaluation of the current $ProcMCE(C, T, D)$ procedure. In other words, d is the size of the maximum set of vertices $X \subseteq T$ that can be added to C to obtain a maximal clique. Let $G_T = (V_{G_T}, E_{G_T})$ be the subgraph of G induced by T. Thus, X is a maximum clique in G_T and hence to compute X exactly is NP-hard (with G_T as input). Therefore, we devise heuristics to upper bound $|X|$. We discuss three heuristics as follows.

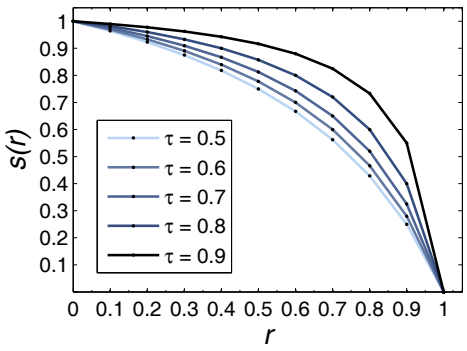

Figure 4: Sampling probability $s(r)$ for $0.5 \leq \tau \leq 0.9$

Let $\overline{d}_{vnum}$ be the maximum degree of any vertex in the subgraph G_T, $\overline{d}_h$ be the maximum value of h so that there are h vertices with degree at least h in G_T, and $\overline{d}_{core}$ be the maximum core number of G_T (i.e., the maximum value of c such that every vertex in a subgraph F of G_T has degree at least c within F and F is the largest such subgraph of G_T). It is easy to see that $\overline{d}_{vnum}$, $\overline{d}_h$, and $\overline{d}_{core}$ are all upper bounds of $|X|$ since X forms a complete subgraph in G_T. By their definition, we can order the upper bounds as follows.

$$\overline{d}_{vnum} \geq \overline{d}_h \geq \overline{d}_{core} \geq |X|$$

The smaller the value of the upper bound, the greater is the pruning power. On the other hand, it requires

$$O(1) \leq O(|V_{G_T}|) \leq O(|E_{G_T}|)$$

time to compute the bounds, respectively. Thus, there is a tradeoff between pruning power and efficiency. The effect of the choice of the upper bound is evaluated later in our experiments.

EXAMPLE 3. *Here we consider how the lower bound $\tilde{r}$ is determined in Algorithm 2 and Algorithm 3 for graph G in Figure 3(a). When $ProcMCE(\{3,2\}, \{1,7\}, \phi)$ is processed, the current subtree $\mathcal{T}$ contains 2 candidate vertices: $T = \{1, 7\}$. Thus we estimate $\overline{d} = 2$. At this point, $C = \{3, 2\}$. Hence the size of any clique to be generated from $\mathcal{T}$ is at most $|C| + 2$=4. If we do not consider future intersection with existing cliques, then the current overlap with existing clique is only $C \cap C'$ where $C' = \{3, 6, 4, 1\}$ and $\tilde{r}$ is given by $1/4 = 0.25$. To tighten the lower bound, we consider possible future intersection of P with existing cliques. If we let $|P| = 4$, then $t = 2$. Consider the previous maximal clique $C' = \{3, 6, 4, 1\}$ in $\mathcal{S}$. $Y = T \setminus C' = \{1, 7\} - \{3, 6, 4, 1\} = \{7\}$. Hence $y_t = 1$. To determine a lower bound $\tilde{r}$, we choose 7 to be the next vertex in the potential P before we choose 1. Hence the ratio $(|C \cap C'| + \max\{t - \overline{y_t}, 0\})/(|C| + t) = \frac{1+1}{4} = 0.5$. If we let $|P| = 3$, then $t = 1$, for the worst case, we choose 7 to be the final vertex in P, and $(|C \cap C'| + \max\{t - \overline{y_t}, 0\})/(|C| + t) = \frac{1+0}{3} = 0.33$. $\tilde{r} = \min_{1 \leq t \leq \overline{d}} \frac{|C \cap C'| + \max\{t - \overline{y_t}, 0\}}{|C| + t} = 0.33$.* □

To estimate $\overline{y_t}$, we may also use $|Y|$ similarly, or the number of vertices in Y with degree at least t, or the number of vertices in Y.

Algorithm 3: Deterministic Summarization

Input: Graph G, maximum redundancy τ
Output: A τ-visible summary $\mathcal{S}$ of $\mathcal{M}(G)$

1 $\mathcal{S} \leftarrow \emptyset$;
2 Let C' be the maximal clique previously included in $\mathcal{S}$;
3 $C' \leftarrow \emptyset$;
4 Call $ProcMCE(\emptyset, V, \emptyset)$, during the process when entering each recursion of $ProcMCE(C, T, D)$, process Lines 5-9:
5 Compute $\tilde{r}$ by applying Lines 5-7 of Algorithm 2 here ;
6 **if** $\tilde{r} > \tau$ **then**
  // prune $\mathcal{T}$
7 **return**;
8 **end**
9 Execute $ProcMCE(C, T, D)$ if $\mathcal{T}$ is not pruned, for each C computed to be a maximal clique, do: $\mathcal{S} = \mathcal{S} \cup C, C' \leftarrow C$

A perspective of Algorithm 2. To get an intuition of our algorithm, recall the notion of importance sampling as used in the approximation of integral of a (unknown) continuous function, for example, $y = f(x)$. The idea is to draw more samples in regions where y changes rapidly and less where the curve is relatively flat. Analogously, we can imagine a y-value being a maximal clique C and the corresponding x-value being the time-stamp when C is outputted. By our observation, the output sequence of maximal cliques appears "continuous" where sharp changes are infrequent. Outputting maximal cliques with probabilities adaptive to the changes (large coverage ratio indicating a small change) then approximates the entire space of maximal cliques.

Derandomization. We now describe a method to derandomize Algorithm 2 in order to compute an exact τ-visible summary. The algorithm also leverages the local similarity between maximal cliques in the output sequence of classic MCE to achieve effective pruning, as outlined in Algorithm 3.

Each time when the recursive procedure $ProcMCE(C, T, D)$ is starting a new search subtree $\mathcal{T}$, we first examine if the search in $\mathcal{T}$ is capable of leading us to any maximal clique that is sufficiently different from the maximal clique previously included in the summary $\mathcal{S}$ so far. We use the techniques similar to those used for the randomized algorithm to compute $\tilde{r}$. Then, $\mathcal{T}$ is always discarded when $\tilde{r} > \tau$ and always retained when $\tilde{r} \leq \tau$.

The deterministic algorithm always summarizes $\mathcal{M}(G)$ with a guarantee in terms of visibility, since a maximal clique C is not included in $\mathcal{S}$ only when a fraction of at least τ of C is covered by $C' \in \mathcal{S}$. In this case, the visibility of C is ensured by the presence of C'. The result is summarized in Theorem 2 as follows.

THEOREM 2. *Algorithm 3 computes a τ-visible summary.*

4.2 Global Filtering

Although the redundancy removal using local similarity can already significantly reduce the output size of classic MCE, we can further remove redundancy with a global view of all maximal cliques in the summary.

In Algorithm 4, when a maximal clique C is found by procedure $ProcMCE$, C is first tested for redundancy with the existing maximal cliques in the current $\mathcal{S}$. It is costly to compare C with every $C' \in \mathcal{S}$. We devise the following method to allow efficient redundancy testing for every newly found maximal clique with the maximal cliques in $\mathcal{S}$.

Let f_X and b_X be the smallest and greatest id of any vertex in a clique X. Then, the vertex ids of X lie in the range $[f_X, b_X]$. Instead of comparing C with every $C' \in \mathcal{S}$, we retrieve C' only

Algorithm 4: Summarization with Global Filtering

Input: Graph G, maximum redundancy τ
Output: A τ-visible summary $\mathcal{S}$ of $\mathcal{M}(G)$

1 $\mathcal{S} \leftarrow \emptyset$;
2 Let $\mathcal{L}$ be a binary-search tree containing maximal cliques in current $\mathcal{S}$ that may potentially be compared with future generated maximal cliques;
3 $\mathcal{L} \leftarrow \emptyset$;
  // i is the vertex id of v_i
4 Let vertices in V in ascending order of their id be $v_1, v_2, ..., v_n$;
5 $T = V, D = \emptyset$;
6 **for** $i \leftarrow 1$ **to** n **do**
  // Let b_C denote the greatest id of any vertex in a clique C
7 Update $\mathcal{L}$ by removing every C' from $\mathcal{L}$ if $b_{C'} < i$;
8 $T \leftarrow T \setminus \{v\}$;
9 Call $ProcMCE(\{v_i\}, T \cap \mathcal{N}(v_i), D \cap \mathcal{N}(v_i))$;
10 $D \leftarrow D \cup \{v\}$;
11 In the recursive calls of $ProcMCE$, search subtrees are pruned by techniques in Algorithm 2 or 3, and Lines 13-19 below are executed whenever a maximal clique C is found;
12 **end**
13 **foreach** *maximal clique C found by $ProcMCE$* **do**
14 Compare C to each clique in $\mathcal{L}$;
15 **if** *no $C' \in \mathcal{L}$ exists such that* $|C' \cap C|/|C| > \tau$ **then**
16 $\mathcal{S} \leftarrow \mathcal{S} \cup C$;
17 Insert C into $\mathcal{L}$ with b_C as the key and C as the value;
18 **end**
19 **end**

when $[f_C, b_C] \cap [f_{C'}, b_{C'}]$ is nonempty. To find such C' quickly, we maintain a binary-search tree $\mathcal{L}$ with $b_{C'}$ as the key and $C' \in \mathcal{S}$ as the value. Then, each relevant C' can be retrieved in $O(\log |\mathcal{S}|)$ time.

In addition, we devise a method so that we do not keep every $C' \in \mathcal{S}$ in $\mathcal{L}$ as follows. Assume in the first level recursion of $ProcMCE$, we pick a vertex in ascending order of the vertex id. Thus, a new clique C is generated in ascending order of f_C. In this way, it is safe to remove a maximal clique C' from $\mathcal{L}$ if $b_{C'} < f_C$, since C' cannot overlap with any maximal cliques found later.

To show the correctness of the algorithm we use the following lemma [21].

LEMMA 1 ([21]). *$ProcMCE(C, T, D)$ return all and only maximal cliques containing all vertices in C, some vertices in T, and no vertex in D, without duplication.*

THEOREM 3. *Algorithm 4 returns a τ-visible summary $\mathcal{S}$.*

PROOF. Let C be a maximal clique and v_c be the vertex with the smallest id in C. When $ProcMCE(\{v_c\}, T_c, D_c)$ is called in Algorithm 4, by Lemma 1 all and only the maximal cliques containing v_c and no vertex in $\{v_1, ..., v_{c-1}\}$ will be generated. Since $ProcMCE(C, T, D)$ is called by the algorithm for $C = \{v_i\}$ for each $v_i \in V$, each maximal clique of G will be generated exactly once or pruned by Algorithm 2 or 3 for being found redundant. Thus, $\mathcal{S}$ is a τ-visible summary since Lines 15-18 make sure that C is not included into $\mathcal{S}$ only when $|C' \cap C|/|C| > \tau$ for an existing C' in the current $\mathcal{S}$. □

A heuristic can be applied to shorten the interval $[f_C, b_C]$ so that C can be discarded earlier. Initially, G is clustered and the vertices are assigned id's in a way that vertices in the same cluster have close id's. Since vertices in a clique is highly interconnected, their id's also tend to cluster and fall in a relatively small interval.

5. APPLICATIONS

The results of τ-visible summary can be used for different applications. Here we consider two possible refinement processes: top-k enumeration based on diversity and interactive clique exploration. Both processes can help end users to further sharpen their result sets and will be valuable in the investigation of different types of graph data.

5.1 Top-k Maximal Cliques with Diversity

Top-k computation is a useful and important operation in database systems and related applications, especially when the size of results is massive making it difficult to use the results. We consider computing the top-k results, $\mathcal{A}$, of $\mathcal{M}(G)$. To ensure the quality of $\mathcal{A}$, we should not rely only on the quality of any individual result in $\mathcal{A}$, but also be aware of the relationship between them. In particular, we desire results in the top-k answers to be *diverse*.

While quality measures of $\mathcal{A} \subseteq \mathcal{M}(G)$ with diversity consideration depend on applications, most of these measures are NP-hard to compute. However, an interesting family of evaluation functions, called submodular functions, allow efficient approximation of the top-k results.

A set function $f : 2^N \rightarrow \mathbb{R}$ is submodular if and only if for all sets $S, T \subseteq N$ such that $S \subseteq T$, and $d \in N \backslash T$, $f(S + d) - f(S) \geq f(T + d) - f(T)$ [19]. For example, in the NP-hard problem of maximum k-set coverage problem (MkC), the function that evaluates the coverage of a collection of sets can be shown to be submodular.

We discuss efficient top-k clique computation with any submodular measure, using the objective function in MkC as an example. That is, we compute a collection $\mathcal{A}$ of k maximal cliques that covers the maximum number of vertices in G. With some $\mathcal{M}' \subseteq \mathcal{M}(G)$ as the input, the algorithm each time greedily selects a maximal clique $C \in \mathcal{M}'$ that adds maximum marginal coverage to the current $\mathcal{A}$ and include C in $\mathcal{A}$. From the result by Nemhauser, Wolsey, and Fisher [19], the resulting collection has a coverage that is not smaller than $(1 - 1/e)$ times the optimal solution. Thus, our algorithm is $(1 - 1/e)$-approximate and has complexity $O(k\alpha|\mathcal{M}'|)$, where α is the average clique size in $\mathcal{M}'$.

However, even if we can compute $\mathcal{A}$ efficiently by taking advantage of a submodular measure, the computation can still be impractically expensive with the input $\mathcal{M}(G)$ from a classic MCE algorithm, due to the explosion in the number of maximal cliques. With τ-visible MCE, however, the problem can be readily solved with a significantly smaller τ-visible summary $\mathcal{S}$ as the input, while the quality of $\mathcal{A}$ given $\mathcal{S}$ is still comparable to that by $\mathcal{M}(G)$ as verified by our experiments.

5.2 Interactive Clique Exploration

We propose *Interactive Clique Exploration* (ICE) that is natural and efficient for a user to incrementally approach the target clique. On the contrary, a classic MCE algorithm can overwhelm the users with too many results, thus severely hindering the applications of maximal cliques. The workflow of ICE is depicted in Figure 5.

Initially we select a set of seed vertices Σ from V, which means that we are to grow cliques containing at least a vertex from Σ. Let G^+_Σ be the *extended subgraph* of Σ, i.e., G^+_Σ is the subgraph of G induced by the set of vertices containing Σ and all neighbors of Σ. We then apply τ-visible MCE to G^+_Σ to compute a τ-visible summary $\mathcal{S}$ of $\mathcal{M}(G^+_\Sigma)$. The list of top-k answers $\mathcal{A}(\mathcal{S})$ in $\mathcal{S}$ is then computed by the greedy algorithm described in Section 5.1, which is efficient with a submodular quality measure and $\mathcal{S}$ as input.

Next, $\mathcal{A}$ is presented to the user. When the user finds some $C^* \in \mathcal{A}$ to be the target, ICE terminates with C^* as the answer.

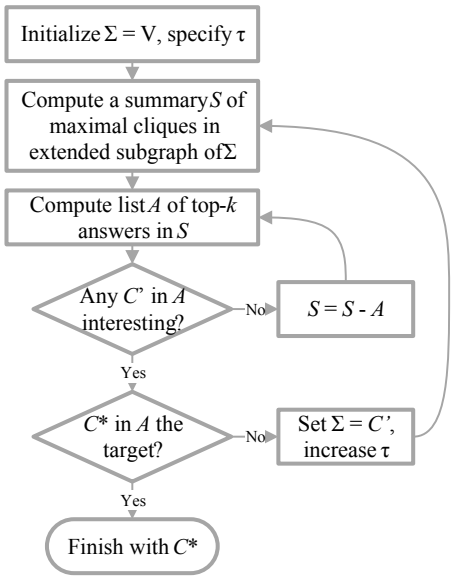

Figure 5: Workflow of ICE

Otherwise, if the user finds some $C' \in \mathcal{A}$ interesting and it worths a further exploration on C', we set $\Sigma = C'$ and increase τ to zoom in the vicinity of C', and start from Step 2 for a new iteration now with $G^+_{C'}$. In case $\mathcal{A}(\mathcal{S})$ is unsatisfactory, $\mathcal{A}(\mathcal{S})$ is removed from $\mathcal{S}$ and we re-compute the top-k answers in the new summary $\mathcal{S}$, and the new $\mathcal{A}$ is again presented to the user as shown in Figure 5.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate the efficiency and effectiveness of our method, compared with the classic MCE algorithm. For short, we denote by **MCE** the classic backtracking MCE algorithm [21] that has been proved to achieve optimal worst-case complexity, by **τ-RMCE** the randomized algorithm that computes an expected τ-visible summary, and by **τ-DMCE** the deterministic algorithm that computes an exact τ-visible MCE. The experiments were run on a machine with Intel Core i5 3.30GHz CPU and 4GB main memory.

Datasets. We use four real world graphs to evaluate the algorithms. Some statistics about the datasets are given in Table 1. *Blog* is from the blogs network and has vertices as blogs and an edge indicates that two blogs appear in the same search result of the top-15 popular queries published by Technorati. *Skitter* describes an internet topology constructed from several sources to about a million destinations. *Wiki* represents Wikipedia users as vertices and an edge indicates that a user once edited a talk page of another user. *Patent* is a graph with U.S. patents as vertices and citations between them as edges.

The default setting in our experiments is $\tau = 0.8$ and using $\overline{d}_h$ as the estimation heuristic for $\overline{d}$ (see details in Section 4.1).

6.1 Classic MCE v.s. τ-Visible MCE

We demonstrate the advantage of τ-visible MCE algorithms, τ-RMCE and τ-DMCE, over the classic MCE algorithm, MCE, by comparing their running time and output size. We vary τ from 0.5 to 1 (note that we choose $\tau >= 0.5$ for practical visibility insurance).

The running time of τ-RMCE and τ-DMCE is reported in Figure 6, where the running time for MCE is equivalent to the time at

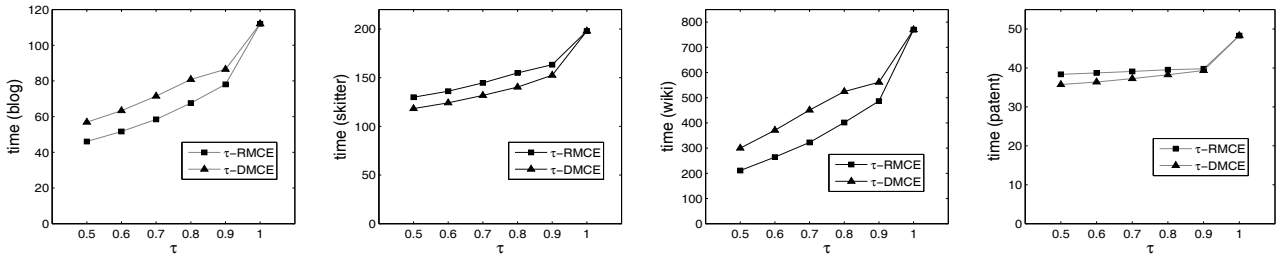

Figure 6: Running time (in seconds) of τ-RMCE and τ-DMCE with varying τ, and of MCE (at $\tau = 1$)

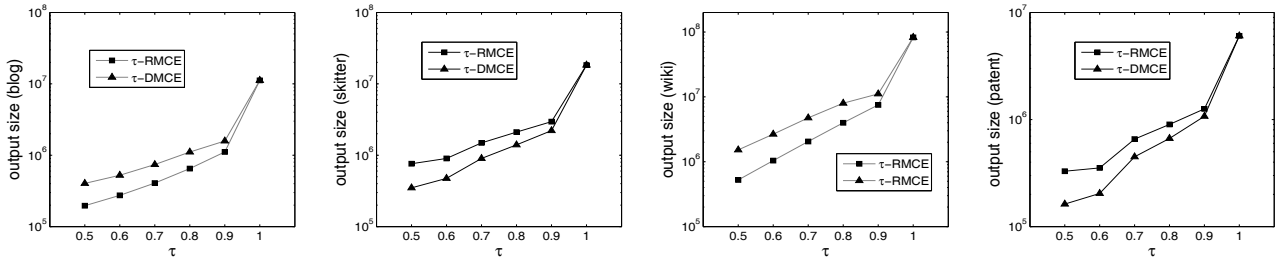

Figure 7: Size of the τ-visible summary by τ-RMCE and τ-DMCE with varying τ, and the total number of maximal cliques outputted by MCE (at $\tau = 1$)

Table 1: Statistics of datasets (K $= 10^3$, M $= 10^6$): the number of vertices and edges ($|V|$ and $|E|$), the size of maximum clique and average size of maximal cliques (c_{max} and c_{avg}), and total number of maximal cliques with size at least 3 ($|\mathcal{M}|$) in graph G

	Blog	Skitter	Wiki	Patent
$\|V\|$	990K	1.7M	2.39M	3.7M
$\|E\|$	6.62M	11.1M	41.7M	33M
c_{max}	49	67	26	11
c_{avg}	15	21	13	6
$\|\mathcal{M}\|$	11.2M	18.3M	82.7M	6.1M

$\tau = 1$. The result shows that, for all the datasets, the running time drops sharply when τ changes from $\tau = 1$ to $\tau = 0.9$, which indicates a significant reduction in time with only a small loss of completeness. The result also shows a clear trend of decreasing running time as τ ranges from 0.9 to 0.5.

Figure 7 reports the size of the τ-visible summary computed by τ-RMCE and τ-DMCE, respectively, where we also report the total number of maximal cliques returned by MCE (i.e., at $\tau = 1$) as a reference. The result verifies our claim in the significant level of redundancy in the set of maximal cliques. The number of maximal cliques decreases exponentially when τ changes from $\tau = 1$ to $\tau = 0.9$, and the number continues to decrease significantly when τ changes from 0.9 to 0.5. In other words, the redundancy already begins to drop dramatically as we start to introduce the τ-visible summary with a small change in the visibility threshold τ.

6.2 Choice of Estimation Heuristics

The estimation of $\overline{d}$ can affect the performance of τ-RMCE and τ-DMCE considerably, as shown by the result using τ-RMCE in Table 2 (the result for τ-DMCE is similar as the same principle is applied). While the core number can provide the tightest bound $\overline{d}_{core}$, it results in much longer running time. On the contrary, the heuristics $\overline{d}_{vnum}$ and $\overline{d}_h$ result in significantly better efficiency. These two heuristics are no more costly than the set intersection operations in the $ProcMCE$ procedure, thus giving the same time complexity and incurring only little time overhead. Thus, the tighter bound $\overline{d}_h$, though having slightly higher overhead than computing the simple heuristic $\overline{d}_{vnum}$, turns out to improve the efficiency of τ-RMCE more than the low-overhead $\overline{d}_{vnum}$.

Table 2: Running time (in seconds) of τ-RMCE with $\overline{d}$ estimated by $\overline{d}_{vnum}$, $\overline{d}_h$, and $\overline{d}_{core}$

	Blog	Skitter	Wiki	Patent
$\overline{d}_{vnum}$	71.4	151.1	459.8	43.9
$\overline{d}_h$	66.7	140.2	405.4	37.9
$\overline{d}_{core}$	>3600	>3600	>3600	>3600

Table 3: Running time (in seconds), t_{local}/t_{global}, of τ-RMCE and the number of maximal cliques in the τ-visible summary, $|\mathcal{S}_{local}|/|\mathcal{S}_{global}|$, with/without global filtering

	Blog	Skitter	Wiki	Patent
t_{local}	67.6	140.3	401.5	38.3
t_{global}	72.8	142.6	1132	38.9
$\|\mathcal{S}_{local}\|$	650K	1.40M	3.98M	665.8K
$\|\mathcal{S}_{global}\|$	601K	1.37M	3.31M	666.5K

6.3 Refinement with Global Filter

The global filter described in Section 4.2 further refines the τ-visible summary $\mathcal{S}$ computed by τ-RMCE and τ-DMCE. Its effect in producing a smaller summary and the additional time overhead are evaluated by the results using τ-RMCE in Table 3 (the results using τ-DMCE are similar).

As reported in Table 3, the running time for the datasets Blog, Skitter and Patent is only marginally increased with the addition of global filter, but it takes 3 times more time to finish on Wiki.

As for the size of the τ-visible summary $\mathcal{S}$, we observe that the reduction in the number of maximal cliques by a global filter is not as obvious as compared with those filtered locally (as shown in Figure 7). This agrees with our observation of local similarity between maximal cliques in the output sequence of classic recursive backtracking MCE algorithms.

6.4 Performance of Top-k Computation

Finally, we use top-k maximal clique retrieval as an example to demonstrate how τ-visible MCE can be useful in practice. The objective function in MkC (maximum k-set coverage) as discussed in Section 5.1 is used as the quality measure. In our experiment, we set $k = 20$, as a typical setting in information retrieval applications. We report in Table 4 the comparison of the quality of top-k results and the time taken to compute them from a τ-visible summary and from the set of all maximal cliques in the graph.

Table 4: Quality of and running time (in seconds) to compute top-20 results from the τ-visible summary by τ-RMCE (Q_{rand}, T_{rand}) and τ-DMCE(Q_{det}, T_{det}), and from the set of all maximal cliques (Q_{all}, T_{all})

	Blog	Skitter	Wiki	Patent
Q_{rand}	822	1205	462	173
Q_{det}	826	1214	464	174
Q_{all}	907	1255	485	195
T_{rand}	1.38	4.02	8.59	0.70
T_{det}	2.41	4.97	20.25	1.00
T_{all}	27.42	47.51	196.95	8.87

As shown in Table 4, the quality (i.e., the number of vertices in the graph being covered by the top-20 results) of the top-20 results computed from a τ-visible summary is very close to that computed from the set of all maximal cliques. However, computing top-20 results from a τ-visible summary is approximately an order of magnitude faster than from the set of all maximal cliques.

The results thus support the use of a τ-visible summary for top-k computation for its much smaller size (resulting in higher efficiency) without sacrificing the quality.

7. CONCLUSIONS

In this paper, we highlighted the problem of excessive size and redundancy in classic maximal clique enumeration (MCE). We introduced the notion of τ-visible MCE to reduce the redundancy while capturing the major information in the result. We proposed efficient algorithms for the computation of a τ-visible summary, and introduced top-k clique computation on top of the summary to further enhance the result usability. Our empirical studies showed significant improvements in both the result size and computation time over classic MCE, and demonstrated the usefulness of a τ-visible summary.

8. REFERENCES

[1] J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN*, pages 598–612, 2002.

[2] F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *SIGKDD*, 2004.

[3] R. Agrawal, S. Gollapudi, A. Halverson, and S. Leong. Diversifying search results. In *Second ACM Int. Conf. on Web Search and Data Mining (WSDM)*, 2009.

[4] N. M. Berry, T. H. Ko, T. Moy, J. Smrcka, J. Turnley, and B. Wu. Emergent clique formation in terrorist recruitment. In *The AAAI-04 Workshop on Agent Organizations: Theory and Practice*, 2004.

[5] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.

[6] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theor. Comput. Sci.*, 407(1-3):564–568, 2008.

[7] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD Conference*, pages 447–458, 2010.

[9] 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):21, 2011.

[10] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *KDD*, pages 1240–1248, 2012.

[11] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *KDD*, pages 672–680, 2011.

[12] S. Chu and J. Cheng. Triangle listing in massive networks. *TKDD*, 6(4):17, 2012.

[13] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC (1)*, pages 403–414, 2010.

[14] M. Hua, J. Pei, A. Fu, and X. Lin. Top-k typicality queries and efficient query answering methods on large databases. *The VLDB Journal*, 18:809–835, 2009.

[15] F. Kose, W. Weckwerth, T. Linke, and O. Fiehn. Visualizing plant metabolomic correlation networks using clique-metabolite matrices. *Bioinformatics*, 17(12):1198–1208, 2001.

[16] G. Liu and L. Wong. Effective pruning techniques for mining quasi-cliques. In *ECML/PKDD (2)*, pages 33–49, 2008.

[17] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci.*, 210(2):305–325, 1999.

[18] J. Moon and L. Moser. On cliques in graphs. *Israel J. Math*, 3:23–28, 1965.

[19] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Math. Programming*, 14:265–294, 1978.

[20] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *SIGKDD*, 2005.

[21] 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):28–42, 2006.

[22] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.

[23] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[24] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *SIGKDD*, 2006.

[25] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. In *SIGKDD*, 2005.