Efficient \((k, \alpha)\)-Maximal-Cliques Enumeration over Uncertain Graphs

JING BAI\(^1\), JUNFENG ZHOU\(^2\), MING DU\(^3\), PENG ZHONG\(^4\)

School of Computer Science Technology, Donghua University, Shanghai, China (e-mail: 18016297367@163.com; {zhoujf,duming,zhongpeng}@dhu.edu.cn)
\(^1\)College of Information Technology, Shanghai Jian Qiao University, Shanghai, China (e-mail: 18016297367@163.com)

Corresponding author: Jing Bai (e-mail: 18016297367@163.com), Junfeng Zhou (e-mail: zhoujf@dhu.edu.cn).

This work was partly supported by the grants from the Natural Science Foundation of Shanghai (No. 20ZR1402700), and from the Natural Science Foundation of China (No.: 61472339, 61572421, 61272124).

\section{Abstract}
A maximal clique (MC) is a complete subgraph satisfying that no other cliques can take it as their proper subgraph. Given an uncertain graph, the top-\(K\) MCs enumeration problem studies how to return \(k\) MCs with the highest rank value. Existing algorithms rank MCs according to their probabilities, thus usually return MCs with higher probabilities but less number of vertices, and fail to return large MCs that convey more useful information. Considering this problem, this paper studies the problem of enumerating top-\(K\) MCs. Our approach returns \(k\) MCs with the most number of vertices satisfying that their probabilities \(\geq\alpha\), where each MC is called an \(\alpha\)-MC, and computing \(k\) largest \(\alpha\)-MCs is called \((k,\alpha)\)-MCs. We propose an efficient \((k,\alpha)\)-MCs enumeration algorithm, Top-\(\alpha\)-KMC, which works in three steps, including partition, enumeration and verification. Here, partition means that we compute the set \(\mathcal{M}\) of all MCs without considering the probability information, as if the graph is partitioned into a set of subgraphs. Enumeration means that we compute \(\alpha\)-MCs from each MC of \(\mathcal{M}\). As each such subgraph is an MC, the cost of computing common neighbors for finding \(\alpha\)-MCs can be reduced. Verification means that we need to verify whether an \(\alpha\)-MC is a subgraph of another \(\alpha\)-MC. If not, it is an \(\alpha\)-MC; otherwise, it is a useless \(\alpha\)-MC and should be removed. We further propose an optimized algorithm Top-\(\alpha\)-KMC+ to reduce both time and space by merging the above three steps into a whole step. The experimental results on real datasets show that both Top-\(\alpha\)-KMC and Top-\(\alpha\)-KMC+ can return \(k\) largest \(\alpha\)-MCs efficiently.

\section{Index Terms}
Uncertain Graph, Maximal Clique, top-\(K\) Maximal Clique

\section{Introduction}

Graphs have been widely used to describe the complex relationships between different entities. In practice, due to noise [1], measurement errors [2], the accuracy of predictions [3], [4] and privacy concerns [5], these relationships do not exist definitely, such as protein-protein interaction (PPI) networks with experimentally inferred links [6]–[8], social networks with inferred influence [9], [10], and sensor networks with uncertain connectivity links [11]. In these applications, we often model the uncertain relationships using an uncertain graph, where the existence of each edge is denoted by a probability [12].

A maximal clique (MC) in a graph \(G\) is a complete subgraph satisfying that no other cliques can take it as their proper subgraph [13]–[16]. MC enumeration has attracted significant attention [17]–[20], due to its numerous applications in real-world problems, such as finding overlapping communities from social networks [21]–[27], recommending products for sales [28], [29], detecting ratings [30], [31], discovering followers on social networks, and identifying protein complexes in PPI networks [6]–[8], etc. MC enumeration in uncertain graph is used to find from the underlying graph all MCs satisfying that each one has probability \(\geq\alpha\), where \(\alpha\) is a threshold specified by users [17], and each MC is called an \(\alpha\)-MC. However, \(\alpha\)-MC enumeration is time-consuming since the number of MCs can be exponential in the number of vertices [17], which makes the enumeration computationally intractable. Furthermore, it is not necessary to enumerate all the \(\alpha\)-MCs but some largest ones, i.e., top-\(K\) \(\alpha\)-MCs, which convey sufficient and valid information.

Most existing works focus on returning top-\(K\) MCs on deterministic graphs [32]–[34]. For uncertain graphs, however, to the best of our knowledge, only [18] and [35] have tried to study the problem of top-\(K\) \(\alpha\)-MCs. The authors proposed to return \(k\) \(\alpha\)-MCs satisfying that they have the largest probabilities and the number of vertices is not less than a given threshold. As they rank \(\alpha\)-MCs according to their probabilities, they usually return \(\alpha\)-MCs with higher probabilities but less number of vertices, and fail to return large \(\alpha\)-MCs that convey more useful information. For example, for the uncertain graph in Figure 1, when enumerating the top-1 \(\alpha\)-
MCs according to their probabilities by limiting that the number of vertices \( \geq 3 \), we can obtain the top-1 \( \alpha \)-MCs \( \{v_7, v_{10}, v_{11}\} \) with the largest probabilities. However, users may be more interested in the MC \( \{v_5, v_6, v_7, v_8\} \) which has the largest sizes. Considering this problem, we propose to return \( k \) largest MCs satisfying that their probabilities \( \geq \alpha \), which is denoted as \((k, \alpha)\)-MCs. For example, if using our approach and limiting \( \alpha = 0.1 \), then the result of top-1 \( \alpha \)-MCs w.r.t. Figure 1 is \( \{v_5, v_6, v_7, v_8\} \).

**Challenge.** Since an MC with more vertices may have the clique probability less than \( \alpha \), and an MC with a larger clique probability may have less number of vertices, we must compute all the \( \alpha \)-MCs to find the \((k, \alpha)\)-MCs. For example, in Figure 1, when we set the probability threshold \( \alpha = 0.7 \), based on the vertex ID, the first produced \( \alpha \)-MC is \( \{v_1, v_2\} \), which is small in fact. For this graph, the last produced \( \alpha \)-MC \( \{v_7, v_{10}, v_{11}\} \) is the largest one with clique probability \( \geq \alpha \). We propose a baseline approach which returns the \((k, \alpha)\)-MCs by first enumerating all the \( \alpha \)-MCs, then sorting them by the number of vertices, such that to get the result of \((k, \alpha)\)-MCs. Therefore, the efficiency of \( \alpha \)-MC enumeration will dominate the overall performance. During computing an \( \alpha \)-MC starting from any vertex in the original graph, we need to expand the current \( \alpha \)-MC by a new vertex \( v \). Here, the basic operation is computing the common neighbors of \( v \) and the vertices in the current \( \alpha \)-MC. Obviously, the less the number of common neighbor computations involved, the higher the efficiency. For example, if we compute \( \alpha \)-MC on \( G' \) in Figure 1, when we set the probability threshold \( \alpha = 0.1 \), we will find three \( \alpha \)-MCs \( \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\} \), which means we have to compute the common neighbors three times for \( \{v_1, v_3\}, \{v_1, v_4\} \), and \( \{v_3, v_4\} \). However, if we know that \( G' \) is an MC, then we do not need to perform any common neighbor computation. To do that, we first compute MCs on \( G \) in Figure 1 to get the \( \text{MCG} \), where we only need to perform the common neighbor computation once to get the \( \text{MCG} \).

**Our approach.** Motivated by the above observation, we propose a novel idea to compute \( \alpha \)-MCs from MCs that are computed first. We first propose an algorithm, Top-KMC, which works in three steps for \( \alpha \)-MCs computation. The first step is the partition step, in which we compute the set \( M \) of all MCs without considering the probability information, and if the graph is divided into a set of subgraphs. The second step is the enumeration step, in which we compute \( \alpha \)-MCs from each MC of \( M \). As each such subgraph is an MC, which is an complete subgraph, the cost of common neighbor computation can be avoided. The last step is the verification step, in which we need to verify whether an \( \alpha \)-MC is a subgraph of another \( \alpha \)-MC. If not, it is an \( \alpha \)-MC; otherwise, it is a pseudo \( \alpha \)-MC and should be removed. Although Top-KMC reduces the number of common neighbor computations, it still suffers from inefficiency in time and space. First, we need to compute \( \alpha \)-MCs over all the MCs to find all the \( \alpha \)-MCs. Second, we need to store all the MCs in \( M \) and all the \( \alpha \)-MCs in result set \( R \).

To make a further improvement, we propose an optimized algorithm Top-KMC+, which combines the three steps of Top-KMC into a whole step, and only maintains a result set of \( k \cdot 2^{l_{\max}} \) to store the largest \( \alpha \)-MCs, where \( l_{\max} \) is the largest truss number[36] of the graph. Specifically, when finding an MC, we immediately compute the \( \alpha \)-MCs over this MC, then we maintain a result set of \( k \cdot 2^{l_{\max}} \) to store the candidate \( \alpha \)-MCs. Finally we only verify at most \( k \cdot 2^{l_{\max}} \) cliques to find \((k, \alpha)\)-MCs.

Our contributions are as follows.

1. We first propose an efficient top-\( K \) MC enumeration algorithm, Top-KMC, which works in three steps, including partition, enumeration and verification, to improve the efficiency.
2. We further propose an optimized algorithm Top-KMC+, which merges the above three steps into a whole step to reduce both time and space.
3. We conduct experiments on 12 uncertain datasets. The experimental results show that both Top-KMC and Top-KMC+ can return \((k, \alpha)\)-MCs efficiently.

**Organization.** The rest of this paper is summarized as follows. Section II provides some preliminaries. In section III, we give a review of existing works. In section V, we propose an efficient algorithm Top-KMC. In section VI, we further introduce and analyze an optimized algorithm Top-KMC+. Section VII shows the experimental results. Section VIII concludes this paper.

### II. PRELIMINARIES

**Definition 2.1:** [Uncertain graph] An uncertain graph is denoted as \( G = (V, E, \beta) \), where \( V \) represents a set of vertices in \( G \), \( E \) a set of edges in \( G \) and \( \beta \) a function that maps each edge in \( E \) to a probability value in \([0, 1]\).

For example, Figure 1 is an uncertain graph.

**Definition 2.2:** [Clique] Given a graph \( G = (V, E) \), the vertex set \( C \subseteq V \) is a clique, if \( C \neq \emptyset \), and there is an edge between each pair of vertices in \( C \).

For example, there are many cliques in Figure 1, such as \( \{v_5, v_6, v_7\}, \{v_5, v_6, v_7, v_8\} \), etc.

For simplicity, we also say the edges connecting vertices of \( C \) as \( C \)'s edges. Obviously, given a clique \( C \) of \( n \) vertices, \( C \) has \( n(n-1)/2 \) edges. We use max(\( C \)) to denote the vertex in \( C \) that has the largest ID.

**Definition 2.3:** [Maximal Clique (MC)] Given a graph \( G = (V, E) \) and clique \( C \subseteq V \), we say \( C \) is an MC, if there is no vertex \( v \in V \setminus C \) such that \( C \cup \{v\} \) is a clique.

For example, in Figure 1, \( \{v_5, v_6, v_7\} \) is a clique but not an MC, due to there is a vertex \( v_8 \) such that \( \{v_5, v_6, v_7\} \cup \{v_8\} \) is a clique, and \( \{v_5, v_6, v_7, v_8\} \) is an MC due to there is no vertex \( v \in V \setminus C \) such that \( \{v_5, v_6, v_7, v_8\} \cup \{v\} \) is a clique.

**Definition 2.4:** [Clique size] Given a clique \( C \), the size of \( C \) is the number of vertices in \( C \), denoted as size(\( C \)).

For example, \( C_1 = \{v_5, v_6, v_7, v_8\} \) is larger than \( C_2 = \{v_5, v_6, v_7\} \), since size(\( C_1 \)) = 4 > size(\( C_2 \)) = 3.

**Definition 2.5:** [Clique probability] Given an uncertain graph \( G = (V, E, \beta) \), for any clique \( C \) in \( G \), the clique probability \( clq(C) \) is the product of the probabilities of all edges in \( C \).

For example, in Figure 1, let \( C = \{v_1, v_2, v_3\} \), then \( clq(C) = 0.36 \).
Definition 2.6: [α-clique] Given an uncertain graph $G = (V, E, \beta)$ and a probability threshold $\alpha$, $C$ is an $\alpha$-clique, if it satisfies (1) $C$ is a clique; and (2) $clq(C) \geq \alpha$.

For example, in Figure 1, if $\alpha = 0.1$, then $C_1 = \{v_1, v_2, v_3\}$ and $C_2 = \{v_2, v_3\}$ are $\alpha$-cliques because $clq(C_1) = 0.36 \geq 0.1$ and $clq(C_2) = 0.8 \geq 0.1$, but $C_3 = \{v_1, v_3, v_4\}$ is not an $\alpha$-clique because $clq(C_3) = 0.05 < 0.1$.

Definition 2.7: [$\alpha$-MC] Given an $\alpha$-clique $C$, we say $C$ is an $\alpha$-MC, if there is no $v \in V \setminus C$ such that $C \cup \{v\}$ is an $\alpha$-clique.

For example, in Figure 1, if $\alpha = 0.1$, then $C_1 = \{v_1, v_2, v_3\}$ is an $\alpha$-MC, but $C_2 = \{v_1, v_2\}$ is a pseudo-$\alpha$-MC because $C_2 \cup \{v_3\}$ is an $\alpha$-clique.

Definition 2.8: [(k, $\alpha$)-MCs] Given an uncertain graph $G = (V, E, \beta)$, a probability threshold $\alpha$, and an integer $k$, we use $(k, \alpha)$-MCs to denote the set of $k$ largest $\alpha$-MCs.

For example, in Figure 1, if $\alpha = 0.1$ and $k = 3$, all the $\alpha$-MCs are $\{v_3, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_7, v_9\}$, and the $(k, \alpha)$-MCs are $\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}$, and $\{v_7, v_{10}, v_{11}\}$.

Definition 2.9: [Support [36]] Given a graph $G = (V, E)$, the support of an edge $e(u, v) \in E$, denoted by $sup_{e(u,v)}$, is defined as $|\Delta_{uvw}|$, where $\Delta_{uvw}$ denotes the triangle of three vertices $u, v, w \in V$. $sup_{e(u,v)} = |N_G(u) \cap N_G(v)|$.

Definition 2.10: [k-truss [36]] Given a graph $G = (V, E)$, the $k$-truss of $G(k \geq 2)$, denoted by $T_k$, is defined as the largest subgraph of $G$ such that for any edge $e \in E$, denoted by $t_e$, is defined as the maximum $k$ of the $k$-truss that the edge $e$ is in.

Observation 2.1: Based on Definition 2.10, given an edge $e \in E_G$, for each $k$-truss $T_k$ containing $e$, $t_e = \max(k)$.

For example, in Figure 1, $sup_{e(v_1, v_2)} = 1, sup_{e(v_2, v_3)} = 1, sup_{e(v_1, v_3)} = 2, sup_{e(v_1, v_4)} = 1$ and $sup_{e(v_3, v_4)} = 1$. Given $k = 3$, by Observation 2.1, the truss numbers $t_e(v_1, v_2) = 3, t_e(v_2, v_3) = 3, t_e(v_1, v_3) = 3, t_e(v_1, v_4) = 3$ and $t_e(v_3, v_4) = 3$.

Problem Statement: Given an uncertain graph $G = (V, E, \beta)$, a probability threshold $\alpha$, and an integer $k$, return $(k, \alpha)$-MCs.

III. RELATED WORKS

We discuss related work from four aspects according to whether we are computing all the MCs and whether the given graph is a deterministic graph, as in below.

MCs enumeration on deterministic graphs. The classic MCs enumeration algorithm on deterministic graphs is the Bron-Kerbosch algorithm [37], which is based on DFS approach. In 2006, Tomita et al. [38] proposed an improved algorithm BKwith a better strategy for pivot selection based on Bron-Kerbosch, which runs in time $O(3^{V/3})$. In 2011, D. Eppstein et al. [39] further modified Bron-Kerbosch by choosing more carefully the order in which the vertices are processed, which runs in time $O(3^{2V/3})$ with $n$ vertices and degeneracy $d$.

MCs enumeration on uncertain graphs. In 2015, Pan et al. [17] proposed an algorithm MULE on uncertain graphs, which computes all $\alpha$-MCs by processing vertices in ascending order of their IDs. In 2019, Ahmar Rashid et al. [20] proposed an approach EMCTDS to enumerate $\alpha$-MCs. EMCTDS utilizes a novel index named $h$-index to maintain vertices with degree greater than $h$. EMCTDS first finds the $\alpha$-MCs with a descending order in sizes then tracks and deletes all the subsets of the clique. In 2019, Rong-Hua Li et al. [19] proposed two core-based pruning algorithms to reduce the graph size before computing $\alpha$-MCs, such that to accelerate the computation.

Top-$K$ MCs enumeration on deterministic graphs. In 2015, Yuan et al. [32] proposed an efficient top-$K$ MCs enumeration algorithm EnumK for deterministic graphs. EnumK processes vertices in the order based on DFS, and obtains the top-$K$ MCs by maintaining a result set of $k$. In 2018, Apurba et al. [33] first proposed a concept of Maximal Quasi-clique, which requires that the degree of vertices in this subgraph should be greater than a given threshold. Then they proposed an algorithm TopMaximalQC for the top-$K$ MCs enumeration by pruning the vertices with a small core-number, such that to reduce redundant computations. In 2020, Wu et al. [34] proposed an efficient algorithm TOPKLS for finding top-$K$ MCs, which maintains a result set of $k$ that can be dynamically adjusted to facilitate pruning small cliques.

Top-$K$ MCs enumeration on uncertain graphs. The concept of probability top-$K$ MCs in uncertain graphs is first proposed by Zou et al. [18] in 2010. Zhu Rong et al. [35] further studied how to
compute top-$K$ $\alpha$-MCs on large-scale uncertain graphs. Both [18] and [35] return $k$ $\alpha$-MCs that have the largest probabilities and the sizes are not less than a given threshold, due to that they rank $\alpha$-MCs according to their probabilities. As a result, they may return $\alpha$-MCs with higher probabilities but less number of vertices, and fail to return large MCs that convey more useful information.

In practice, users may be more interested in $\alpha$-MCs with the largest sizes, since they convey more useful information. Our work takes this into consideration and returns $k$ largest $\alpha$-MCs with probabilities $\geq \alpha$.

IV. BASELINE APPROACH

A. ALGORITHM IDEA

Since an MC with more vertices may have the clique probability less than $\alpha$, and an MC with a larger clique probability may have less number of vertices, it is difficult to find the $(k, \alpha)$-MCs before knowing all the $\alpha$-MCs. The baseline approach, i.e., Algorithm 1, returns the $(k, \alpha)$-MCs by first enumerating all the $\alpha$-MCs (line 4), then sorting them by their clique sizes (line 5), such that to get $(k, \alpha)$-MCs (line 6).

In Algorithm 1, we enumerate all the $\alpha$-MCs by processing the vertices over the uncertain graph in ascending order according to their IDs. During the processing, we maintain three sets $C$, $I$, and $X$ to recursively compute $\alpha$-MCs, where $C$ is used to store the current $\alpha$-MC, $I$ is used to store the candidate vertices that are connected to each vertex in $C$ for expanding the current $\alpha$-MC, and $X$ is used to store the vertices that are connected to each vertex in $C$ but definitely will not be added to $C$ for ensuring each $\alpha$-MC is enumerated only once. Both $I$ and $X$ are dynamically updated by Functions GenerateI and GenerateX when $C$ is expanded (lines 14 to 15). Note that, adding a new vertex $v$ to $C$ will decrease the clique probability of $C$ by a probability equal to the product of the edges probabilities between $v$ and each vertex in $C$. Thus, we need to compute the clique probability of current $C$ before adding the new vertex for ensuring the clique probability of $C$ is no less than $\alpha$. In Function GenerateI, each tuple $(u, v)$ stored in $I$ satisfies that $u$ is the common neighbor of all vertices in $C$, $u > \max(C)$ (line 22) and $\text{clq}(C \cup \{u\}) \geq \alpha$ (line 24), where $\alpha$ denotes the probability that the current $\text{clq}(C)$ will multiply when vertex $u$ is added to $C$. In Function GenerateX, each tuple $(u, v)$ stored in $X$ satisfies that $u$ is the common neighbor of all vertices in $C$ and $\text{clq}(C \cup \{u\}) \geq \alpha$ (lines 27 to 33). The expansion will terminate when both $I$ and $X$ are empty (line 8).

Example 1: Considering Figure 1. Given $\alpha = 0.1$ and $k = 3$, we enumerate $\alpha$-MCs starting from the first vertex $v_1$. The three set $C$, $I$, and $X$ are updated dynamically. The running status of $C$, $I$ and $X$ are shown in Table 1. When both $I$ and $X$ are empty, it means that the current $C$ is an $\alpha$-MC. When processing vertices $v_1, v_2, v_3$ and $v_4$ in ascending order of their IDs, we get four $\alpha$-MCs, $\{v_1, v_2, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}$ and $\{v_1, v_2, v_3, v_4\}$. At the end of the whole enumeration, we find all the $\alpha$-MCs: $\{v_1, v_2, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_1, v_2, v_3, v_4\}, \{v_1, v_2, v_7, v_8\}, \{v_7, v_9\}$ and $\{v_7, v_{10}, v_{11}\}$. Then we sort them according to their sizes, and output the first 3 $\alpha$-MCs: $\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}$ and $\{v_7, v_{10}, v_{11}\}$.

Algorithm 1: Baseline $(k, \alpha)$-MCs Enumeration

Input: Graph $G = (V, E, \beta)$, integer $k$ and a probability threshold $\alpha$
Output: $(k, \alpha)$-MCs

1. $I \leftarrow \emptyset$
2. foreach $u \in V$ do
   2.1. $I' \leftarrow I \cup \{(u, 1)\}$
   3. $R \leftarrow \text{EUMC}(\emptyset, 1, I', \emptyset)$
   4. Sort cliques in $R$ according to their sizes
   5. output first $k$ $(\alpha)$-MCs of $R$
   7. Function EUMC($C$, $q$, $I$, $X$)
   8. if ($I = \emptyset$ and $X = \emptyset$) then
      9. $R \leftarrow \text{EUMC}(C', q', I'$, $X'$)
   10. else
      11. foreach $(u, r) \in I$ do
         12. $C' \leftarrow C \cup \{u\}$
         13. $q' \leftarrow q \cdot r$
         14. $I' \leftarrow \text{GenerateI}(C', q', I)$
         15. $X' \leftarrow \text{GenerateX}(C', q', X, r)$
         16. $R \leftarrow \text{EUMC}(C', q', I', X')$
         17. $X \leftarrow X \cup \{(u, r)\}$
      18. return $R$
   19. Function GenerateI($C'$, $q'$, $I$)
   20. $m \leftarrow \max(C'), I' \leftarrow \emptyset$
   21. foreach $(u, r) \in I$ do
      22. if $u > m$ then
         23. $\text{clq}(C') \leftarrow \text{clq}(C' \cup \{u\})$
         24. if $\text{clq}(C') \geq \alpha$ then
            25. $I' \leftarrow I' \cup \{(u', r')\}$
   26. return $I'$
   27. Function GenerateX($C'$, $q'$, $X$)
   28. $m \leftarrow \max(C'), X' \leftarrow \emptyset$
   29. foreach $(u, r) \in X$ do
      30. if $u > m$ then
         31. $\text{clq}(C') \leftarrow \text{clq}(C' \cup \{u\})$
         32. if $\text{clq}(C') \geq \alpha$ then
            33. $X' \leftarrow X' \cup \{(u', r')\}$
   34. return $X'$

B. ANALYSIS OF ALGORITHM

From Algorithm 1 we know that the running time of Function EUMC dominates the whole process. An execution of the recursive Function EUMC can be viewed as a search tree [17]. Each call to EUMC is a node of this search tree. The first call to the method is the root node. A node in this search tree is either an internal node that makes one or more recursive calls, or a leaf node that does not make further recursive calls. Specifically, the running time at each leaf node is $O(1)$. This is because there are no further recursive calls, that is, $I$ is empty, and checking the size of $I$ takes constant time. The time taken at each internal node is $O(|V|)$. Line 12 takes $O(|V|)$ time as we add all vertices in $C$ to $C'$ and also $\alpha$. Line 13 takes constant time. Lines 14 and 15 take $O(|V|)$ time. Furthermore, for each vertex, the total number for call Function EUMC is no more than $O(2|V|)$. In conclusion, the time complexity of Algorithm 1 is $O(|V| \cdot 2|V|)$.
TABLE 1: The running status of $C$, I and $X$, where for each tuple $(u, r)$ in I and $X$, we omit the probability $r$ due to space limit.

| $C$          | $I$                      | $X$                      |
|--------------|--------------------------|--------------------------|
| $\emptyset$ | $\emptyset$              | $\emptyset$              |
| $v_1$        | $v_2$, $v_3$, $v_4$      | $\emptyset$              |
| $v_1$, $v_2$ | $v_3$                   | $\emptyset$              |
| $v_1$, $v_2$, $v_3$ | $\emptyset$              | $\emptyset$              |
| $v_1$, $v_3$ | $v_2$                   | $\emptyset$              |
| $v_1$, $v_4$ | $\emptyset$              | $\emptyset$              |
| $v_2$, $v_3$ | $\emptyset$              | $\emptyset$              |
| $v_2$, $v_4$ | $\emptyset$              | $\emptyset$              |
| $v_3$, $v_4$ | $\emptyset$              | $\emptyset$              |
| $\ldots$    | $\ldots$                | $\ldots$                |

Example 3: Consider Example 2. We compute all the $\alpha$-MCs from each $MC$ in $M$ by calling Function EMMC. When computing $\alpha$-MCs from \{v_1, v_2, v_3\}, \{v_3, v_5\}, \{v_5, v_6, v_7, v_8\}, \{v_7, v_9\} and \{v_9, v_{10}, v_1\}, we directly put them into the result set $R$ since their clique probabilities are not less than the probability threshold $\alpha$ (lines 8 to 9). However, when computing \{v_1, v_3, v_4\}, we need to call Function EUMC to get all the $\alpha$-MCs \{v_1, v_3\}, \{v_1, v_4\} and \{v_3, v_4\}, since $clg(\{v_1, v_3, v_4\})$ is less than $\alpha$. The result is $R = \{\{v_1, v_2, v_3\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_1\}\}$.

V. TOP-KMC ALGORITHM

A. ALGORITHM IDEA

From the above discussion we know that the efficiency of $\alpha$-MC enumeration is limited by the frequent common neighbor computations of vertices in the current clique $C$. Obviously, the less the number of common neighbor computations involved, the higher the efficiency. That is, if we first divide the given uncertain graph into some $MC$s, without considering the probability information, we do not need to perform any common neighbor computations when we compute $\alpha$-MCs on these $MC$s. Such preprocessing will help to avoid redundant common neighbor computations since the number of common neighbor computations of $MC$s is much less than that of $\alpha$-MC. Note that, there may be some overlap among different $MC$s, which means that one $\alpha$-MC computed from an $MC$ may be a subgraph of another $\alpha$-MC computed from other $MC$s. We need to remove such $\alpha$-MCs before returning the final results.

Based on this observation, we propose Top-KMC, i.e., Algorithm 2, to compute $\alpha$-MCs in three steps. First, we compute the set $M$ of all $MC$s without considering the probability information in line 1 by calling Function BKPivot[38]. Second, we compute the result set $R$ of all $\alpha$-MCs from each $MC$ in $M$ (line 3). Third, we identify the first $k$ largest $\alpha$-MCs (line 4), and return them (line 5).

Example 2: Considering Figure 1. For the given probability threshold $\alpha = 0.1$ and $k = 5$, we compute all the $MC$s using the BKPivot algorithm, the result is $M = \{\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_3, v_5\}, \{v_5, v_6, v_7, v_8\}, \{v_7, v_9\}, \{v_9, v_{10}, v_1\}\}$. Next, we mainly discuss the details of the second and third steps in below.

Enumeration of $\alpha$-MCs from MC (Function EMMC). In Function EMMC, we compute $\alpha$-MCs from each $MC$ in $M$. As there is an edge between each pair of vertices in an $MC$, the cost of common neighbor computations can be avoided. For each $MC$ $g$, we first check whether $g$’s probability is not less than $\alpha$ (line 8). If so, $g$ is an $\alpha$-MC, then we directly put it into the result set $R$ without further processing (line 9). Otherwise, we compute $\alpha$-MCs from $g$ in line 14 by calling Function EUMC, where we do not need to compute any common neighbors because $g$ itself is an $MC$ (lines 10 to 15).

Example 3: Consider Example 2. We compute all the $\alpha$-MCs from each $MC$ in $M$ by calling Function EMMC. When computing $\alpha$-MCs from \{v_1, v_2, v_3\}, \{v_3, v_5\}, \{v_5, v_6, v_7, v_8\}, \{v_7, v_9\} and \{v_9, v_{10}, v_1\}, we directly put them into the result set $R$ since their clique probabilities are not less than the probability threshold $\alpha$ (lines 8 to 9). However, when computing \{v_1, v_3, v_4\}, we need to call Function EUMC to get all the $\alpha$-MCs \{v_1, v_3\}, \{v_1, v_4\} and \{v_3, v_4\}, since $clg(\{v_1, v_3, v_4\})$ is less than $\alpha$. The result is $R = \{\{v_1, v_2, v_3\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_5, v_6, v_7, v_8\}, \{v_9, v_{10}, v_1\}\}$.

Fast Removing Pseudo-$\alpha$-MCs (Function FRPMC). There may exist some overlaps between different $MC$s from Function BKPivot, which means that one $\alpha$-MC computed from an $MC$ may be a subgraph of another $\alpha$-MC computed from other $MC$s. Therefore, before returning the final results, we need to remove these pseudo-$\alpha$-MCs, which are subgraphs of another $\alpha$-MC. To identify whether an $\alpha$-clique in $R$ is a pseudo-$\alpha$-MC, the intuitive yet expensive way is comparing it with all other $\alpha$-cliques in $R$ to check whether it is a subgraph of another $\alpha$-clique. Our approach makes acceleration using the following observation.

Observation 5.1: Given two cliques $C_1$ and $C_2$, if $C_1$ is a subgraph of $C_2$, then size($C_1$) ≤ size($C_2$).

By Observation 5.1, we first sort all $\alpha$-cliques in $R$ according to their sizes in descending order (line 17), then compare each $\alpha$-clique with the ones before it to find the top-$K$ results. However, when we process the $n^{th}$ $\alpha$-clique, we need to compare it with the $n - 1$ other $\alpha$-cliques before it, which means that we need to check whether it is a subgraph of another one $n - 1$ times. For further accelerating the process, we propose an inverted index based approach to identify the pseudo-$\alpha$-MCs. We assign each vertex an inverted list to record which $\alpha$-MCs it belongs to (line 18). This inverted list is constructed during processing the $\alpha$-cliques in $R$. Let $C$ be the $n^{th}$ processed $\alpha$-clique in $R$ in descending order w.r.t. the clique sizes, if $C$ is an $\alpha$-MC, then we add $n$ to every inverted list of the vertices in $C$. The basic idea is to transform multiple subgraph checking operations into one set intersection operation (line 21). We have the following result to guarantee its correctness.

Theorem 1: Assume that all the $\alpha$-cliques are processed in descending order w.r.t. their sizes, and each vertex $v$ is associated with an inverted list $L[v]$ recording which cliques it belongs to. Given an $\alpha$-clique $C$, $C$ is an $\alpha$-MC iff $S = \bigcap_{v \in C} L[v]$ is not empty, $S = \emptyset$.

Proof: Given an $\alpha$-clique $C$, if the set intersection for inverted lists of $C$ is an empty set, it means that there does not exist a larger $\alpha$-MC containing all the vertices in $C$. Therefore, $C$ is not a subgraph of any other $\alpha$-MCs, that is, $C$ is an $\alpha$-MC.

On the other hand, if $C$ is an $\alpha$-MC, it means that it is not a subgraph of any other $\alpha$-cliques in $R$. And before processing $C$, its processing order has not yet been added to its inverted lists. Therefore, if $C$ is an $\alpha$-MC, the set intersection of its inverted lists is empty.

Based on Theorem 1, when processing an $\alpha$-clique $C$, we just need to perform set intersection on inverted lists of $C$ (line 21). If the
result is an empty set, it means that $C$ is not a subgraph of any other $\alpha$-cliques before $C$, then we know immediately that $C$ is an $\alpha$-MC (lines 24 to 30); otherwise, $C$ is a pseudo-$\alpha$-MC and we directly delete it from $R$(lines 22 to 23).

**Example 4:** Continue Example 3. Given all the $\alpha$-cliques in $R$, we need to identify and remove all the pseudo-$\alpha$-MCs to get the final results. We first sort all the $\alpha$-cliques in $R$ according to their sizes. The result is $R = \{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_9\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_7, v_9\}\}$. Then we verify these $\alpha$-cliques by dynamically updating inverted lists of each vertex as shown in Table 2. Specifically, when processing the first $\alpha$-clique $\{v_5, v_6, v_7, v_8\}$, we compute the set intersection of inverted lists of the four vertices(line 21) and the result is an empty set, because each inverted list has been initialized as an empty set at the beginning of Function FRPMC. According to Theorem 1, it is an $\alpha$-MC, then we update the inverted lists of $v_5, v_6, v_7$ and $v_8$ as $L(v_5) = \{1\}$, $L(v_6) = \{1\}$, $L(v_7) = \{1\}$ and $L(v_8) = \{1\}$ (lines 24 to 30), as the second column shown. Next, we process the second $\alpha$-clique $\{v_1, v_2, v_3\}$. Since the set intersection of inverted lists of $v_1, v_2$ and $v_3$ is an empty set, it means that $\{v_1, v_2, v_3\}$ is an $\alpha$-MC, then we update the inverted lists of vertex $v_1, v_2$ and $v_3$ as $L(v_1) = \{2\}$, $L(v_2) = \{2\}$ and $L(v_3) = \{2\}$, as shown in the third column. Similarly, for the third $\alpha$-clique $\{v_7, v_{10}, v_{11}\}$, the result of set intersection is empty, which means $\{v_7, v_{10}, v_{11}\}$ is an $\alpha$-MC. We update the the inverted lists of vertex $v_7, v_{10}$ and $v_{11}$ as $L(v_7) = \{1, 3\}$, $L(v_{10}) = \{3\}$ and $L(v_{11}) = \{3\}$. For the forth $\alpha$-clique $\{v_1, v_3\}$, the set intersection of inverted lists of $v_1$ and $v_3$ is $S=L(v_1) \cap L(v_3)=\{2\} \neq \emptyset$, which means that both $v_1$ and $v_3$ belong to the second $\alpha$-MC $\{v_1, v_2, v_3\}$, thus $\{v_1, v_3\}$ is a subgraph of $\{v_1, v_2, v_3\}$. Then we delete $\{v_1, v_3\}$ (lines 22 to 23). Similarly, both the fifth $\alpha$-clique $\{v_1, v_4\}$ and the sixth $\alpha$-clique $\{v_3, v_4\}$ are $\alpha$-MCs. Hereafter, we have got the first five $\alpha$-MCs, Function FRPMC is terminated without further processing. Finally, the result set is $\{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_4\}, \{v_3, v_4\}\}$, given that $\alpha = 0.1$ and $k = 5$.

### B. ANALYSIS

From Algorithm 2, we know that the whole time complexity is composed of three parts, which are Functions BKPIvot, EMMC and FRPMC.

Given a graph $G$, the worst time complexity of BKPIvot(line 1) is $O(3^{|V|^2/3})$, and $G$ can be divided into at most $3^{|V|^2/3}$ MCs [38].

| Vertex | First iteration | Second iteration | ... | Final iteration |
|--------|----------------|-----------------|-----|----------------|
| $v_1$  | $\emptyset$   | 2               | ... | 2, 4           |
| $v_2$  | $\emptyset$   | 2               | ... | 2             |
| $v_3$  | $\emptyset$   | 2               | ... | 2, 5           |
| $v_4$  | $\emptyset$   | $\emptyset$    | ... | 4, 5           |
| $v_5$  | 1              | 1               | ... | 1             |
| $v_6$  | 1              | 1               | ... | 1             |
| $v_7$  | 1              | 1               | ... | 1             |
| $v_8$  | 1              | 1               | ... | 1             |
| $v_9$  | $\emptyset$   | $\emptyset$    | ... | $\emptyset$   |
| $v_{10}$ | $\emptyset$ | $\emptyset$ | ... | 3 |
| $v_{11}$ | $\emptyset$ | $\emptyset$ | ... | 3 |

**Algorithm 2: Top-KMC**

**Input:** Graph $G = (V, E, \beta)$, integer $k$ and a probability threshold $\alpha$

**Output:** $(k, \alpha)$-MCs

1. $M \leftarrow$ BKPIvot($G$)
2. $R \leftarrow \emptyset$
3. EMMC($M, \alpha, R$)
4. FRPMC($R$)
5. output first $k \alpha$-MCs of $R$
6. Function EMMC($M, \alpha, R$)

7. foreach $g \in M$ do
8. if $clq(g) \geq \alpha$ then
9. $R \leftarrow R \cup \{g\}$
10. else
11. $I \leftarrow \emptyset, A \leftarrow \emptyset$
12. foreach $u \in g$ do
13. $I \leftarrow I \cup \{u\}$
14. $A \leftarrow EUMC(\emptyset, 1, I, \emptyset)$
15. $R \leftarrow R \cup A$

16. Function FRPMC($R$)
17. Sort cliques in $R$ in descending order w.r.t. their sizes
18. Let $L[1, ..., |V|]$ be $|V|$ inverted lists w.r.t. cliques
19. $n \leftarrow 0$
20. foreach $C \in R$ do
21. $S \leftarrow L[v_1] \cap L[v_2] \cap \ldots \cap L[v_{size(C)}]$
22. if $S \neq \emptyset$ then
23. delete $C$ from $R$
24. else
25. $n \leftarrow n + 1$
26. if $n = k$ then
27. return
28. else
29. foreach $v \in C$ do
30. $L[v] \leftarrow L[v] \cup \{n\}$

In Function EMMC(lines 6 to 15), for each MC from BKPIvot, we consider the time spent at the MCs whose clique probabilities are no less than the threshold $\alpha$ as lines 8 and 9 shown, and the other MCs as lines 10 to 15 shown. The runtime at lines 8 and 9 takes constant time to put the MCs with clique probabilities no less than $\alpha$ into the result set $R$. For the other MCs, lines 11 to 14 initialize variables and call Function EUMC.

**Theorem 2:** Given all MCs in $M$, the upper bound for their sizes is $l_{max} = \max(t_e)$, where $e \in E_G$ and $t_e$ is the truss number of $e$.

**Proof:** Given the MCC with the maximum size in $M$, for each $e(u, v) \in C$, there are at least $size(C) - 2$ common neighbors for $u$ and $v$, which means that there are at least $size(C) - 2$ triangles containing $e$. Based on Definition 2.9 and 2.10, we know that $size(C) \leq t_e, \forall e \in C$. Thus, for all MCs in $M$, the upper bound for their sizes is $max(t_e) = l_{max}$.

The time complexity of Function EUMC in Algorithm 1 is $O(|V| \cdot 2^{|V|})$ [17]. Since the input of Function EUMC is an MC
computed by Function BKPivot, based on Theorem 2, the time complexity of Function \(EUMC\) is \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3})\). Note that, \(EUMC\) will be called at most \(3^{V/3}\) times because \(G\) can be divided into at most \(3^{V/3}\) MCs by BKPivot. The time complexity of EMMC is \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3})\).

For Function FRPMC, first, we need to sort all \(\alpha\)-cliques in \(R\). Here, we have counted the number of cliques in some real datasets and find that there are very few large \(\alpha\)-MCs, and their sizes are concentrated in a few numbers, as shown in Table 3. Thus, we sort \(R\) by counting sort. Line 27 takes \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3})\) time by counting sort because there are \(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3}\) \(\alpha\)-cliques to be sorted.

From Theorem 2, we know that the maximum size of each clique \(C\) is bounded by \(t_{\text{max}}\). We only update the inverted lists for the vertices in \(\alpha\)-MCs which means the sizes of \(L\) are bounded by \(k\) (lines 24 to 30). For each clique \(C\) in \(R\), line 21 takes \(O(k \cdot t_{\text{max}})\) time for set intersection.

**Theorem 3:** Given the result set \(R\), after sorting all cliques in \(R\) in descending order according to their sizes, we only need to check the first \(k \cdot 2^{t_{\text{max}}}\) cliques to find all the \((k, \alpha)\)-MCs from \(R\).

**Proof 3:** Given an MCC, the number of \(\alpha\)-cliques of \(C\) is no more than \(2^{t_{\text{max}}}\), where \(\text{size}(C)\) is bounded by \(t_{\text{max}}\) based on Theorem 2. Since we return the top \(k\) results, we only need to check the first \(k \cdot 2^{t_{\text{max}}}\) cliques to find all the \((k, \alpha)\)-MCs from \(R\).

Based on Theorem 3, we need to execute line 21 at most \(k \cdot 2^{t_{\text{max}}}\) times (lines 25 to 27), which means that in the worst case the cost of lines 20 to 30 is \(O(k \cdot t_{\text{max}} \cdot 2^{t_{\text{max}}})\). Thus, the whole time complexity of Function FRPMC is \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3} + k^2 \cdot t_{\text{max}} \cdot 2^{t_{\text{max}}})\). However, \(O(3^{V/3}) \gg O(k^2)\), the time complexity of Function FRPMC is \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3})\).

Based on the above analysis, the whole time complexity of Algorithm 2 is \(O(t_{\text{max}} \cdot 2^{t_{\text{max}}} \cdot 3^{V/3})\).

**VI. TOP-KMC+ ALGORITHM**

**A. ALGORITHM IDEA**

From Example 2, 3 and 4 we know that Top-KMC suffers from large storage space. Specifically, in the first step, we need to maintain a set \(M\) to store all the MCs computed by Function BKPivot. In the second step, we need to maintain all the \(\alpha\)-MCs computed by Function EMMC from each MC in \(M\). However, we only need to verify at most \(k \cdot 2^{t_{\text{max}}}\) \(\alpha\)-cliques in the last step. The reason for this problem is that the three steps are performed separately.

Based on this observation, we propose Top-KMC+, i.e., Algorithm 3, which combines the three steps into one step to compute \(\alpha\)-MCs. First, we use a priority queue \(R\) for maintaining the \(\alpha\)-cliques in descending order according to their sizes (line 1). Here, we maintain \(R\) based on radix sorting, where we put the cliques with the same size into one list, because the clique sizes are concentrated in a few numbers as Table 3 shown. Such processing will take constant time instead of \(O(|\log|R|)\) time by the heap sorting. For a given uncertain graph \(G = (V, E, \beta)\), we compute all MCs without considering the probability information by calling Function BKPivot. Different from Top-KMC, when an MC \(g\) is output by Function BKPivot, we immediately compute \(\alpha\)-MCs form \(g\) by Function ENUMERATE (lines 2 to 3), such that we do not need to maintain an additional set \(M\) to store all the MCs computed by Function BKPivot, where \(G\) can be divided into at most \(3^{V/3}\) MCs. From Theorem 3 we know that, we only need to use the set \(R\) to maintain the largest \(k \cdot 2^{t_{\text{max}}}\) \(\alpha\)-cliques. After getting all the \(k \cdot 2^{t_{\text{max}}}\) \(\alpha\)-cliques, we call Function FRPMC to find all the \((k, \alpha)\)-MCs from \(R\) (lines 4 to 5).

Function ENUMERATE is used to maintain the largest \(k \cdot 2^{t_{\text{max}}}\) \(\alpha\)-cliques computed so far. Specifically, if \(|R|\) is less than \(k \cdot 2^{t_{\text{max}}}\), we directly add the new generated \(\alpha\)-clique into \(R\) (lines 8 to 10); otherwise, we need to check whether each new generated \(\alpha\)-clique is larger than the smallest clique \(C^*\) in \(R\). If \(\text{size}(C) > \text{size}(C^*)\), then \(C^*\) will replace \(C^*\) (line 12); otherwise, \(C^*\) is discarded directly.

**Example 5:** Consider Figure 1, given \(k = 5\) and \(\alpha = 0.1\). First, we use a priority queue \(R \cdot 2^{t_{\text{max}}} = 5 \cdot 2^4 = 80\) to maintain the \(\alpha\)-cliques. Compared to Algorithm 2, which maintains a result set \(R\) of \(2^{V/1} = 2048\) in the worst case, Algorithm 3 saves large storage space. When we get the first MCC \(g_1 = \{v_1, v_2, v_3\}\) from Function BKPivot, Function ENUMERATE computes \(\alpha\)-MCs from \(g_1\). In Function ENUMERATE, since the current \(|R| < k \cdot 2^{t_{\text{max}}} = 5 \cdot 2^4 = 80\), we directly compute \(\alpha\)-MCs and add each new generated \(\alpha\)-MC into Runtill \(|R| = 80\), when \(|R| = 80\) we will replace the new generated \(\alpha\)-MCC with the smallest one \(C^*\) if \(\text{size}(C) > \text{size}(C^*)\). Then the result set \(R = \{v_1, v_2, v_3\}\). Next, similar to the above process, after computing \(\alpha\)-MCs from the second MCC \(g_2 = \{v_1, v_3, v_4\}\), we get \(R = \{v_1, v_2, v_3, \{v_1, v_3, v_4\}, \{v_3, v_4\}, \{v_1, v_4, v_3\}\}\), where \(R\) is a priority queue with counting sort. Similarly, when the last \(\alpha\)-MC \(\{v_7, v_{10}, v_1\}\) is computed from the last MCC \(g_7, v_7, v_{10}, v_1\), we get the final \(R = \{v_5, v_6, v_7, v_8, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_1, v_3\}, \{v_1, v_4, v_3\}, \{v_3, v_4\}, \{v_7, v_{10}, v_1\}\}\). Finally, we call Function FRPMC to verify the result set \(R\) to remove the pseudo-\(\alpha\)-MCs, and output the first \(k \cdot 2^{t_{\text{max}}}\) MCs of \(R\) \(\{v_5, v_6, v_7, v_8, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_1\}\}\) .

**B. ANALYSIS**

From Algorithm 3, we know that the whole time complexity is composed of three parts, which are Function BKPivot, ENUMERATE and FRPMC.

Based on previous section, there are at most \(3^{V/3}\) MCs comput-
A. DATASETS

We used 12 datasets to evaluate the performance of the three algorithms. The datasets include: anthra\textsuperscript{1}, mtrbrv\textsuperscript{1}, agrocyc\textsuperscript{1}, ecoo\textsuperscript{1}, vchocyc\textsuperscript{3}, amaze\textsuperscript{2}, kegg\textsuperscript{3}, xmark\textsuperscript{4}, nasa\textsuperscript{4}, citeseer\textsuperscript{5}, go\textsuperscript{6}, yago\textsuperscript{1}. Among them, anthra, mtrbrv, agrocyc, ecoo, and vchocyc describe the network in biology between the biochemistry and the genome; amaze and kegg describe the network of chemical reactions and neural networks in organisms; xmark and nasa describe the connection network between XML documents; citeseer describes the delivery network of protein reactions; go describes the information between gene ontology; yago describes the semantic information between different semantics relationship. These datasets represent relevant knowledge in different fields. Detailed statistics of these datasets are summarized in Table 4, where $|V|$, $|E|$, $\bar{d}$ and $t_{max}$ represent the number of vertices, the number of edges, the average degree of the maximum truss number in the graph.

B. COMPARISON OF RUNNING TIME

Table 5 to Table 8 show the comparison of running time. We first vary $\alpha$ by fixing $k = 5$, then vary $\alpha$ by fixing $k = 50$, such that to observe the impacts of both $\alpha$ and $k$.

From the comparison between three algorithms, we can see that Top-KMC and Top-KMC+ are more faster than BASIC. This is due to the less the number of common neighbor computations of Top-KMC and Top-KMC+, described in Section V and VI. Specifically, Top-KMC and Top-KMC+ are more than 20 times faster than BASIC on amaze, kegg, xmark, vchocyc, mtrbrv, anthra, ecoo and agrocyc. And Top-KMC and Top-KMC+ are more than 10 times faster than BASIC on nasa and go. However, Top-KMC and Top-KMC+ are only about 10 times faster than BASIC on yago and citeseer. This is because the time complexity of Top-KMC and Top-KMC+ are $O(t_{max} \cdot 2^{t_{max} \cdot 3V/3})$, which

| Table 4: Statistics of real datasets. |
| Dataset | $|V|$ | $|E|$ | $\bar{d}$ | $t_{max}$ |
|----------|-------|-------|-------|--------|
| amaze    | 3,342 | 7,200 | 4.3   | 3      |
| kegg     | 3,615 | 3,908 | 2.2   | 3      |
| nasa     | 5,065 | 6,397 | 2.3   | 3      |
| xmark    | 6,080 | 7,025 | 2.3   | 2      |
| yago     | 6,642 | 42,392| 12.8  | 7      |
| go       | 6,793 | 13,361| 3.9   | 3      |
| vchocyc  | 9,491 | 10,143| 2.1   | 3      |
| mtrbrv   | 9,602 | 10,235| 2.1   | 4      |
| citeseer | 10,120| 44,258| 8.3   | 9      |
| anthra   | 12,495| 26,208| 4.2   | 3      |
| ecoo     | 12,620| 13,350| 2.1   | 3      |
| agrocyc  | 12,684| 26,816| 4.2   | 3      |

\textsuperscript{1}ecocyc.org
\textsuperscript{2}www.amaze.ulb.ac.be
\textsuperscript{3}www.genome.ad.jp/kegg
\textsuperscript{4}docs.baxer.org/wiki/Statistics
\textsuperscript{5}citeseer.ist.psu.edu
\textsuperscript{6}www.genontology.org
\textsuperscript{7}mpi-inf.mpg.de/yago-naga/yago

VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
### TABLE 5: Running time (ms), when $\alpha = 0.1$, $k = 5$.

| Dataset  | BASIC | Top-KMC | Top-KMC+ |
|----------|-------|---------|----------|
| amaze    | 1.956 | 89      | 74       |
| kegg     | 2.019 | 110     | 97       |
| nasa     | 4.087 | 290     | 256      |
| xmark    | 4.832 | 280     | 218      |
| yago     | 8.732 | 951     | 855      |
| go       | 5.677 | 388     | 371      |
| vchocyc  | 10.287| 489     | 459      |
| mtbrv    | 11.142| 497     | 457      |
| citeseer | 13.264| 1,532   | 1,503    |
| anthra   | 18.611| 783     | 750      |
| ecoo     | 19.354| 836     | 795      |
| agrocyc  | 18.906| 813     | 750      |

### TABLE 6: Running time (ms), when $\alpha = 0.5$, $k = 5$.

| Dataset  | BASIC | Top-KMC | Top-KMC+ |
|----------|-------|---------|----------|
| amaze    | 1.634 | 90      | 76       |
| kegg     | 2.019 | 122     | 110      |
| nasa     | 4.140 | 301     | 265      |
| xmark    | 4.724 | 317     | 246      |
| yago     | 6.730 | 1,033   | 896      |
| go       | 5.656 | 410     | 387      |
| vchocyc  | 10.224| 493     | 471      |
| mtbrv    | 10.833| 520     | 470      |
| citeseer | 13.324| 1,589   | 1,501    |
| anthra   | 17.888| 800     | 787      |
| ecoo     | 18.556| 941     | 848      |
| agrocyc  | 18.166| 878     | 758      |

### TABLE 7: Running time (ms), when $\alpha = 0.1$, $k = 5$.

| Dataset  | BASIC | Top-KMC | Top-KMC+ |
|----------|-------|---------|----------|
| amaze    | 1.956 | 91      | 76       |
| kegg     | 2.028 | 114     | 98       |
| nasa     | 4.087 | 300     | 263      |
| xmark    | 4.832 | 315     | 250      |
| yago     | 8.732 | 968     | 897      |
| go       | 5.677 | 389     | 379      |
| vchocyc  | 10.287| 501     | 464      |
| mtbrv    | 11.142| 518     | 467      |
| citeseer | 13.264| 1,598   | 1,570    |
| anthra   | 18.611| 790     | 775      |
| ecoo     | 19.354| 864     | 844      |
| agrocyc  | 18.906| 875     | 751      |

shows that the running time depends largely on $t_{max}$, while the $t_{max}$ of yago and citeseer are larger than the other ones, relatively. Additionally, Top-KMC+ is faster than Top-KMC on all datasets. The reasons are twofold. First, Top-KMC only maintains a result set of $k \cdot 2^{m_{max}}$ such that the time of sorting is reduced. The second is using the size of the smallest clique in the current result set to prune a considerable part of small cliques, as described in Section VI.

**The impact of $\alpha$.** We only report the results for yago and agrocyc, due to the results on other datasets show the similar trends. The running time of Top-KMC+ is shown in Figure 2, when varying $\alpha$ from 0.1 to 0.9 with a given default $k = 100$ on 12 uncertain datasets. According to the experimental results, we can see that with the increase of $\alpha$, the running time of Top-KMC+ is increasing. Because in the Function EMMC, we first check whether the probabilities of the MCs are no less than the threshold $\alpha$. The larger the $\alpha$, the less possibility the MCs satisfy the condition, which means that more MCs need to be further processed by Function EUMC.

**The impact of $k$.** We only report the results for yago and agrocyc, due to the results on other datasets show the similar trends. The running time of Top-KMC+ is shown in Figure 3, when varying $k$ from 10 to 500 with a given default $\alpha = 0.5$, on 12 uncertain datasets. According to the experimental results, we can see that with the increase of $k$, the running time of Top-KMC+ is increasing. Because in Function FRPMC, we update the inverted lists of all the vertices for each $\alpha$-MC until we get the largest $\alpha$-MCs. The larger the $k$, the more update operations we need to do.

### C. COMPARISON OF STORAGE SPACE

We counted the number of cliques and vertices that each algorithm needs to store during the processing, to prove the superiority of Top-KMC+ compared with BASIC and Top-KMC. Recall that BASIC needs to store all the $\alpha$-MCs. Top-KMC+ needs to store all the MCs and $\alpha$-MCs. Top-KMC+ only needs to store first $k \cdot 2^{m_{max}}$ $\alpha$-MCs to find $(k, \alpha)$-MCs. In Table 9, we counted the number of result set $|R|$ storing all the $\alpha$-MCs, the number of set $M$, the result set of $k \cdot 2^{m_{max}}$ of Top-KMC+, and the total number of vertices $|V|$ of all algorithms. From Table 9 we can see that the average space Top-KMC+ needs is about 17 times less than BASIC, and 33 times less than Top-KMC. Especially on the sparse graphs such as xmark, go, vchocyc, anthra, ecoo and agrocyc Top-KMC+ needs space more than 20 times less than BASIC, more than 40 times less than Top-KMC. These results demonstrate that Top-KMC+ needs much less space.
This paper presents two efficient algorithms for enumerating top-$K$ $\alpha$-MCs over uncertain graphs, namely Top-KMC and Top-KMC+. Our algorithms are different from existing algorithms, where existing algorithms rank $\alpha$-MCs according to their probabilities while our algorithms return $k \alpha$-MCs with the most number of vertices satisfying that their probabilities $\geq \alpha$. Top-KMC works in three steps for $\alpha$-MCs computation, which reduces the number of common neighbor computations. Top-KMC+ combines the three steps into one step for reducing the running time and saving more storage space. The experimental results demonstrate the efficiency of Top-KMC and Top-KMC+ both in time and storage space. Especially on sparse graphs (such as yago, kegg, xmark, vchoyc, mtbrv, anthra, ecoo and agrocyc), Top-KMC and Top-KMC+ are on average more than 20 times faster than BASIC. Even on the dense graphs (such as yago and cliseer), Top-KMC and Top-KMC+ are on average about 10 times faster than BASIC. For storage space, the space Top-KMC+ needs is average about 17 times less than BASIC, and 33 times less than Top-KMC.

VIII. CONCLUSION

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
[22] Proceeding of the Second International Multi-Symposium of Computer and Computational Sciences (IMSCCS 2007), August 13-15, 2007, The University of Iowa, Iowa City, Iowa, USA, IEEE Computer Society, 2007.

[23] J. Meena and V. S. Devi, “Overlapping community detection in social network using disjoint community detection,” in IEEE Symposium Series on Computational Intelligence, SSC12015, Cape Town, South Africa, December 7-10, 2015, pp. 764–771, 2015.

[24] L. Bai, X. Cheng, J. Liang, and Y. Guo, “Fast graph clustering with a new description model for community detection,” Inf. Sci., vol. 388, pp. 37–47, 2017.

[25] M. Arab and M. Afsharchi, “Community detection in social networks using hybrid merging of sub-communities,” J. Netw. Comput. Appl., vol. 40, pp. 73–84, 2014.

[26] Newman and E. J. M., “The structure of scientific collaboration networks,” Proceedings of the National Academy of Sciences of the United States of America, 2001.

[27] N. Aldouajj, A. Datta, and J. Li, “Influence propagation model for clique-based community detection in social networks,” IEEE Trans. Comput. Soc. Syst., vol. 5, no. 2, pp. 563–575, 2018.

[28] J. Cheng, Y. Ke, A. W. Fu, J. X. Yu, and L. Zhu, “Finding maximal cliques in massive networks by h*-graph,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010, pp. 447–458, 2010.

[29] W. Chen, Y. Wang, and S. Yang, “Efficient influence maximization in social networks,” in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009, pp. 199–208, 2009.

[30] N. Pathak, S. Mane, and J. Srivastava, “Who thinks who knows who? socio-cognitive analysis of email networks,” in Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China, pp. 466–477, 2006.

[31] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, “FRAUDAR: bounding graph fraud in the face of camouflage,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 895–904, 2016.

[32] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, “Diversified top-k clique search,” in 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015, pp. 387–398, 2015.

[33] S. Sanie-Mehri, A. Das, and S. Tirthapura, “Enumerating top-k quasi-cliques,” in IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018, pp. 1107–1112, 2018.

[34] J. Wu, C. Li, L. Jiang, J. Zhou, and M. Yin, “Local search for diversified top-k clique search problem,” Comput. Oper. Res., vol. 116, p. 104867, 2020.

[35] Z. Zhao-Nian and Z. Rong, “Mining top-k maximal cliques from large uncertain graphs,” in 2013 Chinese Journal of Computers, pp. 2146–2155, CCF, 10 2013.

[36] Y. Che, Z. Lai, S. Sun, Y. Wang, and Q. Luo, “Accelerating truss decomposition on heterogeneous processors,” Proc. VLDB Endow., vol. 13, no. 10, pp. 1751–1764, 2020.

[37] C. Bron and J. Kerbosch, “Finding all cliques of an undirected graph (algorithm 457),” Commun. ACM, vol. 16, no. 9, pp. 575–576, 1973.

[38] E. Tomita, A. Tanaka, and H. Takahashi, “The worst-case time complexity for generating all maximal cliques and computational experiments,” Theor. Comput. Sci., vol. 363, no. 1, pp. 28–42, 2006.

[39] D. Eppstein, M. Löffler, and D. Strash, “Listing all maximal cliques in large sparse real-world graphs,” ACM J. Exp. Algorithmics, vol. 18, 2013.
FIRST A. JING BAI was born in Harbin, Heilongjiang, China, in 1989. She received the B.S. degree in electronic information engineering in 2012 from Northeast Forestry University, and M.S. degree in electronic and communication engineering in 2015 from East China Normal University, Shanghai, China. Since 2019, she has been a doctor student with the School of Computer Science and Technology, Donghua University, Shanghai. Her research interests include information retrieval techniques and optimization on graph data.

SECOND B. JUNFENG ZHOU was born in Xi’an, Shanxi, China, in 1977. He received the B.S. and M.S. degrees in computer science from Yanshan University, in 1999 and 2002, and the Ph.D. degree in computer applications from Renmin University of China, Beijing, China, in 2009. From 2009 to 2017, he was a Professor in Yanshan University. Since 2017, he has been a Professor with the School of Computer Science and Technology, Donghua University, Shanghai. His research interests include information retrieval techniques, query processing on semi-structured data and graph data, and optimization.

THIRD C. MING DU was born in Hulin, Heilongjiang, China, in 1975. He received the B.S. degree in electronic technology in 1998 from Chinese Textile University, and the M.S. degrees in control theory and control engineering in 2005 and the Ph.D. degree in management science and engineering in 2013 from Donghua University, Shanghai, China. He has been an Associate Professor since 2010 in Donghua University, Shanghai. His research interests include information retrieval techniques, Data analysis and mining, and natural language processing.

FOURTH D. PENG ZHONG was born in Wuhan, Hubei, China, in 1997. He received the B.S. degree in automation engineering in 2018 from Wuhan Polytechnic University, and the M.S. degree in computer science from Donghua University in 2020, Shanghai, China. He is a system development and analysis engineer in Bank of Communications Co., Ltd.. His research interests include query processing and optimization on graph data.