IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received September 10, 2021, accepted October 28, 2021, date of publication November 2, 2021, date of current version November 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3125198

Efficient (k, α)-Maximal-Cliques Enumeration Over Uncertain Graphs

JING BAI[1,2], JUNFENG ZHOU[1], MING DU[1], AND PENG ZHONG[1]
[1]School of Computer Science and Technology, Donghua University, Shanghai 201620, China
[2]College of Information Technology, Shanghai Jian Qiao University, Shanghai 201306, China

Corresponding authors: Jing Bai (18016297367@163.com) and Junfeng Zhou (zhoujf@dhu.edu.cn)

This work was supported in part by the Natural Science Foundation of Shanghai under Grant 20ZR1402700; and in part by the Natural Science Foundation of China under Grant 61472339, Grant 61572421, and Grant 61272124.

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 MC*s enumeration problem studies how to return k *MC*s with the highest rank value. Existing algorithms rank *MC*s according to their probabilities, thus usually return *MC*s with higher probabilities but less number of vertices, and fail to return large *MC*s that convey more useful information. Considering this problem, this paper studies the problem of enumerating *top-K MC*s. Our approach returns k *MC*s with the most number of vertices satisfying that their probabilities $\geq \alpha$, where each *MC* is called an α-*MC*, and computing k largest α-*MC*s is called as (k, α)-*MC*s. We propose an efficient (k, α)-*MC*s enumeration algorithm, *Top-KMC*, which works in three steps, including partition, enumeration and verification. Here, partition means that we compute the set $\mathcal{M}$ of all *MC*s without considering the probability information, as if the graph is partitioned into a set of subgraphs. Enumeration means that we compute α-*MC*s from each *MC* of $\mathcal{M}$. As each such subgraph is an *MC*, the cost of computing common neighbors for finding α-*MC*s can be reduced. Verification means that we need to verify whether an α-*MC* is a subgraph of another α-*MC*. If not, it is an α-*MC*; otherwise, it is a useless α-*MC* and should be removed. We further propose an optimized algorithm *Top-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-KMC* and *Top-KMC*+ can return k largest α-*MC*s efficiently.

INDEX TERMS Uncertain graph, maximal clique, *top-K* maximal clique.

I. 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 an uncertain graph is used to find from the underlying graph all *MC*s satisfying that each one has probability $\geq \alpha$, where α is a threshold specified by users [17], and each *MC* is called an α-*MC*. However, α-*MC* enumeration is time-consuming since the number of *MC*s can be exponential in the number of vertices [17], which makes the enumeration computationally intractable. Furthermore, it is not necessary to enumerate all the α-*MC*s but some largest ones, i.e., *top-K* α-*MC*s, which convey sufficient and valid information.

Most existing works focus on returning *top-K MC*s on deterministic graphs [32]–[34]. For uncertain graphs, however, to the best of our knowledge, only [18] and [35] have

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

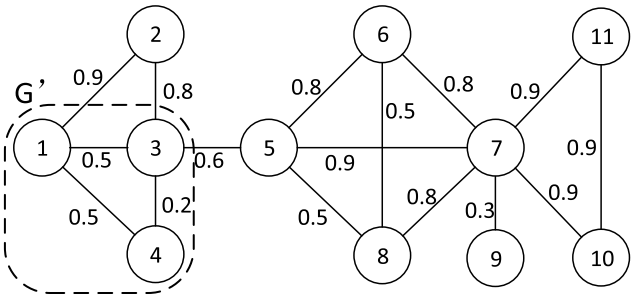


FIGURE 1. Uncertain graph $G = (V, E, \beta)$.

tried to study the problem of *top-K* α-*MC*s. The authors proposed to return k α-*MC*s satisfying that they have the largest probabilities and the number of vertices is not less than a given threshold. As they rank α-*MC*s according to their probabilities, they usually return α-*MC*s with higher probabilities but less number of vertices, and fail to return large α-*MC*s that convey more useful information. For example, for the uncertain graph in Figure 1, when enumerating the top-1 α-*MC*s according to their probabilities by limiting the number of vertices ≥ 3, we can obtain the top-1 α-*MC* $\{v_7, v_{10}, v_{11}\}$ with the largest probability. However, users may be more interested in the *MC* $\{v_5, v_6, v_7, v_8\}$ which has the most vertices. Considering this problem, we propose to return k largest *MC*s satisfying that their probabilities $\geq \alpha$, which is denoted as (k, α)-*MC*s. For example, if using our approach and limiting $\alpha = 0.1$, then the result of top-1 α-*MC* 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 α, and an *MC* with a larger clique probability may have less number of vertices, we must compute all the α-*MC*s to find the (k, α)-*MC*s. For example, in Figure 1, when we set the probability threshold $\alpha = 0.7$, based on the vertex ID, the first produced α-*MC* is $\{v_1, v_2\}$, which is small in fact. For this graph, the last produced α-*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, α)-*MC*s by first enumerating all the α-*MC*s, then sorting them by the number of vertices, such that to get the (k, α)-*MC*s. Therefore, the efficiency of α-*MC* enumeration will dominate the overall performance. During computing an α-*MC* starting from any vertex in the original graph, we need to expand the current α-*MC* by a new vertex v. Here, the basic operation is computing the common neighbors of v and the vertices in the current α-*MC*. Obviously, the less the number of common neighbor computations involved, the higher the efficiency. For example, if we compute α-*MC* on G' in Figure 1, when we set the probability threshold $\alpha = 0.1$, we will find three α-*MC*s $\{v_1, v_3\}$, $\{v_1, v_4\}$, and $\{v_3, v_4\}$, 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 *MC*s on G in Figure 1 to get the *MC* G', where we only need to perform the common neighbor computation once to get the *MC* G'.

Our approach. Motivated by the above observation, we propose a novel idea to compute α-*MC*s from *MC*s that are computed first. We first propose an algorithm, *Top-KMC*, which works in three steps for α-*MC*s computation. The first step is the partition step, in which we compute the set $\mathcal{M}$ of all *MC*s without considering the probability information, as if the graph is divided into a set of subgraphs. The second step is the enumeration step, in which we compute α-*MC*s from each *MC* of $\mathcal{M}$. 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 α-*MC* is a subgraph of another α-*MC*. If not, it is an α-*MC*; otherwise, it is a pseudo α-*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 α-*MC*s over all the *MC*s to find all the α-*MC*s. Second, we need to store all the *MC*s in $\mathcal{M}$ and all the α-*MC*s 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 $k \cdot 2^{t_{max}}$ to store the largest α-*MC*s, where t_{max} is the largest truss number [36] of the graph. Specifically, when finding an *MC*, we immediately compute the α-*MC*s over this *MC*, then we maintain a result set of $k \cdot 2^{t_{max}}$ to store the candidate α-*MC*s. Finally we only verify at most $k \cdot 2^{t_{max}}$ cliques to find (k, α)-*MC*s.

Our contributions are as follows.

1) We first propose an efficient *top-K* α-*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 cost.
3) We conduct experiments on 12 uncertain datasets. The experimental results show that both *Top-KMC* and *Top-KMC+* can return (k, α)-*MC*s 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 donated as $G = (V, E, \beta)$, where V represents a set of vertices in G, E a set of edges in G and β 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 a 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 $\text{size}(C_1) = 4 > \text{size}(C_2) = 3$.

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 α, C is an α-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 α-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 α-clique because $clq(C_3) = 0.05 < 0.1$.

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

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

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

For example, in Figure 1, if $\alpha = 0.1$ and $k = 3$, all the α-MCs are $\{v_5, v_6, v_7, v_8\}$, $\{v_1, v_2, v_3\}$, $\{v_7, v_{10}, v_{11}\}$, $\{v_1, v_4\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$, $\{v_7, v_9\}$, and the (k, α)-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 $|\{\triangle_{uvw} | w \in V\}|$, where $\triangle_{uvw}$ 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 $\forall_{e \in E_{T_k}} sup_e \geq k - 2$. The truss number of an 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 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, $sup_{e(v_1,v_2)} = 1$, $sup_{e(v_2,v_3)} = 1$, $sup_{(v_1,v_3)} = 2$, $sup_{e(v_1,v_4)} = 1$ and $sup_{e(v_3,v_4)} = 1$. Given $k = 3$, by Observation 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 α and an integer k, return (k, α)-MCs.

III. RELATED WORKS

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

MC enumeration on deterministic graphs. The classic MC 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 *BKPivoit* through 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 $O(|V|d3^{d/3})$, where d is the degeneracy.

MC enumeration on uncertain graphs. In 2015, Pan *et al.* [17] proposed an algorithm *MULE* on uncertain graphs, which computes all α-MCs by processing vertices in ascending order of their IDs. In 2019, Ahmar Rashid *et al.* [20] proposed an approach *EMCTDS* to enumerate α-MCs. *EMCTDS* utilizes a novel index named *h_index* to maintain vertices with degrees greater than h. *EMCTDS* first finds the α-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 α-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 MC*s enumeration algorithm *EnumK* for deterministic graphs. *EnumK* processes vertices in the order based on *DFS*, and obtains the *top-K MC*s by maintaining a result set of k. In 2018, Apurba *et al.* [33] first proposed a concept of *MaximalQuasi-clique*, which requires that the degree of vertices in this subgraph should be greater than a given threshold. Then they proposed an algorithm *TopkMaximalQC* for the *top-K MC*s 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 MC*s, 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 *probabilitytop-KMCs* 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* α-*MC*s on large-scale uncertain graphs. Both [18] and [35] return k α-*MC*s that have the largest probabilities and the sizes no less than a given threshold, due to that they rank α-*MC*s according to their probabilities. As a result, they may return α-*MC*s with higher probabilities but less number of vertices, and fail to return large *MC*s that convey more useful information.

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

IV. BASELINE APPROACH

A. ALGORITHM IDEA

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

In Algorithm 1, we enumerate all the α-*MC*s 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 α-*MC*s, where C is used to store the current α-*MC*, I is used to store the candidate vertices that are connected to each vertex in C for expanding the current α-*MC*, and X is used to store the vertices that are connected to each vertex in C but definitely will not to be added to C for ensuring each α-*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 α. In Function *GenerateI*, each tuple (u, r) stored in I satisfies that u is the common neighbor of all vertices in C, $u > \max(C)$(line 22) and $clq(C \cup \{u\}) \geq \alpha$(line 24), where r denotes the probability that the current $clq(C)$ will multiply when vertex u is added to C. In Function *GenerateX*, each tuple (u, r) stored in X satisfies that u is the common neighbor of all vertices in C and $clq(C \cup \{u\}) \geq \alpha$(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 α-*MC*s starting from the first vertex v_1. The three sets 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 α-*MC*. When processing vertices v_1, v_2, v_3 and v_4 in ascending order of their IDs, we get four α-*MC*s, $\{v_1, v_2, v_3\}$, $\{v_1, v_4\}$, $\{v_3, v_4\}$ and $\{v_3, v_5\}$. At the end of the whole enumeration, we find all the α-*MC*s: $\{v_1, v_2, v_3\}$, $\{v_1, v_4\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$, $\{v_5, v_6, v_7, v_8\}$, $\{v_7, v_9\}$ and $\{v_7, v_{10}, v_{11}\}$. Then we sort them according to their sizes, and output the first 3 α-*MC*s: $\{v_5, v_6, v_7, v_8\}$, $\{v_1, v_2, v_3\}$ and $\{v_7, v_{10}, v_{11}\}$.

Algorithm 1: Baseline (k, α)-*MC*s Enumeration

Input: Graph $G = (V, E, \beta)$, integer k and a probability threshold α
Output: (k, α)-*MC*s

1 $I \leftarrow \emptyset$
2 **foreach** $u \in V$ **do**
3 $\quad I' \leftarrow I \cup \{(u, 1)\}$
4 $R \leftarrow EUMC\ (\emptyset, 1, I', \emptyset)$
5 Sort cliques in R according to their sizes
6 **output** first k α-*MC*s of R
7 Function EUMC(C, q, I, X)
8 **if** $(I = \emptyset$ and $X = \emptyset)$ **then**
9 $\quad R \leftarrow R \cup C$
10 **else**
11 $\quad$ **foreach** $(u, r) \in I$ **do**
12 $\quad\quad C' \leftarrow C \cup \{u\}$
13 $\quad\quad q' \leftarrow q \cdot r$
14 $\quad\quad I' \leftarrow GenerateI\ (C', q', I)$
15 $\quad\quad X' \leftarrow GenerateX\ (C', q', X)$
16 $\quad\quad R \leftarrow EUMC\ (C', q', I', X')$
17 $\quad\quad X \leftarrow X \cup \{(u, r)\}$
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 $\quad$ **if** $u > m$ **then**
23 $\quad\quad clq(C') \leftarrow clq(C' \cup \{u\})$
24 $\quad\quad$ **if** $clq(C') \geq \alpha$**then**
25 $\quad\quad\quad I' \leftarrow I' \cup \{(u', r')\}$
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 $\quad clq(C') \leftarrow clq(C' \cup \{u\})$
31 $\quad$ **if** $clq(C') \geq \alpha$**then**
32 $\quad\quad X' \leftarrow X' \cup \{(u', r')\}$
33 **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 of 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

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
$\emptyset$	$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}$	$\emptyset$
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_4	$\emptyset$
v_1, v_3	$\emptyset$	v_2
v_1	v_4	$\emptyset$
v_1, v_4	$\emptyset$	$\emptyset$
v_2	v_3	v_1
v_2, v_3	$\emptyset$	v_1
v_3	v_4, v_5	v_1, v_2
v_3, v_4	$\emptyset$	$\emptyset$
v_3	v_5	v_1, v_2
v_3, v_5	$\emptyset$	$\emptyset$
v_4	$\emptyset$	v_1, v_3
.....	.....	.....

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 u. 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 $O(|V| \cdot 2^{|V|})$.

V. TOP-KMC ALGORITHM

A. ALGORITHM IDEA

From the above discussion we know that the efficiency of α-*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 α-*MC*s on these *MC*s. Such preprocessing will help to avoid redundant common neighbor computations since the number of common neighbor computations of computing *MC*s is much less than that of α-*MC*s. Note that, there may exist some overlaps between different *MC*s, which means that one α-*MC* computed from an *MC* may be a subgraph of another α-*MC* computed from other *MC*s. We need to remove such α-*MC*s before returning the final results.

Based on this observation, we propose *Top-KMC*, ie., Algorithm 2, to compute α-*MC*s in three steps. First, we compute the set $\mathcal{M}$ of all *MC*s without considering the probability information in line 1 by calling Function *BKPivoit* [38]. Second, we compute the result set R of all α-*MC*s from each *MC* in $\mathcal{M}$ (line 3). Third, we identify the first k largest α-*MC*s (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 $\mathcal{M} = \{\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_3, v_5\}, \{v_5, v_6, v_7, v_8\}, \{v_7, v_9\}, \{v_7, v_{10}, v_{11}\}\}$.

Next, we mainly discuss the details of the second and third steps in below.

Enumeration of α-*MC*s from *MC* (Function *EMMC*). In Function *EMMC*, we compute α-*MC*s from each *MC* in $\mathcal{M}$. 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 α(line 8). If so, g is an α-*MC*, then we directly put it into the result set R without further processing(line 9). Otherwise, we compute α-*MC*s from g in line 14 by calling Function *EUMC*, where we do not need any common neighbor computation because g itself is an *MC* (lines 10 to 15).

Example 3: Continue Example 2. We compute all the α-*MC*s from each *MC* in $\mathcal{M}$ by calling Function *EMMC*. When computing α-*MC*s from $\{v_1, v_2, v_3\}$, $\{v_3, v_5\}$, $\{v_5, v_6, v_7, v_8\}$, $\{v_7, v_9\}$ and $\{v_7, v_{10}, v_{11}\}$, we directly put them into the result set R since their clique probabilities are not less than the probability threshold α(lines 8 to 9). However, when computing α-*MC*s from $\{v_1, v_3, v_4\}$, we need to call Function *EUMC* to get all the α-*MC*s $\{v_1, v_3\}$, $\{v_1, v_4\}$ and $\{v_3, v_4\}$, since $clq(\{v_1, v_3, v_4\})$ is less than α. The result is $R = \{\{v_1, v_2, v_3\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_5, v_6, v_7, v_8\}, \{v_7, v_9\}, \{v_7, v_{10}, v_{11}\}\}$.

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

Observation 2: Given two cliques C_1 and C_2, if C_1 is a subgraph of C_2, then size(C_1)$\leq$ size(C_2).

By Observation 2, we first sort all α-cliques in R according to their sizes in descending order(line 17), then compare each α-clique with the ones before it to find the *top-K* results. However, when we process the nth α-clique, we need to compare it with at most $n-1$ α-cliques before it, which means that we need to check whether it is a subgraph of another one at most $n-1$ times. For further acceleration, we propose an inverted index based approach to identify the pseudo-α-*MC*s. We assign each vertex an inverted list to record which α-*MC*s it belongs to(line 18). This inverted list is constructed during processing the α-cliques in R. Let C be the nth processed α-clique in R in descending order w.r.t. the clique sizes, if C is an α-*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 α-cliques are processed in descending order w.r.t. their sizes, and each vertex v is

TABLE 2. Inverted lists of all vertices, $\alpha = 0.1, k = 5$.

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, 3
v_8	1	1	...	1
v_9	$\emptyset$	$\emptyset$	...	$\emptyset$
v_{10}	$\emptyset$	$\emptyset$	...	3
v_{11}	$\emptyset$	$\emptyset$	...	3

associated with an inverted list $L[v]$ recording which cliques it belongs to. Given an α-clique C, C is an α-MC iff $S = L[v_1] \cap L[v_2] \cap \ldots \cap L[v_{size(C)}] = \emptyset$.

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

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

Based on Theorem 1, when processing an α-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 α-cliques before C, then we know immediately that C is an α-MC (lines 24 to 30); otherwise, C is a pseudo-α-MC and we directly delete it from R(lines 22 to 23).

Example 4: Continue Example 3. Given all the α-cliques in R, we need to identify and remove all the pseudo-α-MCs to get the final results. We first sort all the α-cliques in R according to their sizes. The result is $R = \{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_7, v_9\}\}$. Then we verify these α-cliques by dynamically updating inverted lists of each vertex as shown in Table 2. Specifically, when processing the first α-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 α-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 α-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 α-MC, then we update the inverted lists of 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 α-clique $\{v_7, v_{10}, v_{11}\}$, the result of set intersection is empty, which means $\{v_7, v_{10}, v_{11}\}$ is an α-MC, we update the the inverted lists of 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 α-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 α-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 α-clique $\{v_1, v_4\}$ and the sixth α-clique $\{v_3, v_4\}$ are α-MCs. Hereto, we have got the first five α-MCs, Function *FRPMC* is terminated without further processing. Finally, the result set is $R = \{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_4\}, \{v_3, v_4\}\}$, given that $\alpha = 0.1$ and $k = 5$.

Algorithm 2: Top-KMC

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

```
1  M ← BKPivot (G)
2  R ← ∅
3  EMMC (M,α, R)
4  FRPMC (R)
5  output first k α-MCs of R
6  Function EMMC(M, α, R)
7  foreach g ∈ M do
8      if clq(g) ≥ α then
9          R ← R ∪ {g}
10     else
11         I ← ∅, A ← ∅
12         foreach u ∈ g do
13             I ← I ∪ {(u, 1)}
14         A ←EUMC (∅, 1, I, ∅)
15         R ← R∪ 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 ← 0
20 foreach C ∈ R do
21     S← L[v1]∩ L[v2] ∩ . . . ∩ L[v_size(C)]
22     if S≠ ∅ then
23         delete C from R
24     else
25         n ← n + 1
26         if n = k then
27             return
28         else
29             foreach v ∈ C do
30                 L[v] ← L[v] ∪ {n}
```

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 $O(3^{|V|/3})$, and G can be divided into at most $3^{|V|/3}$ MCs [38].

TABLE 3. The number of α-*MC*s in real datasets, when $\alpha = 0.1$.

Dataset	#of α-*MC*s	clique size : #
amaze	3,394	3:112; 2:3,282
kegg	3,680	3:132; 2:3,548
nasa	6,377	3:82; 2:6,295
xmark	7,025	2:7,025
yago	34,957	4:53; 3:20,557; 2:14,347
go	12,333	3:618; 2:12,927
vchocyc	9,893	3:233; 2:9,660
mtbrv	9,995	3:239; 2:9,756
citeseer	38,558	5:3; 4:38,058; 3:18,811; 2:18,740
anthra	12,875	3:198; 2:12,677
ecoo	13,073	3:246; 2:12,827
agrocyc	13,101	3:278; 2:12,823

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

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

Proof: Given the *MCC* with the maximum size in $\mathcal{M}$, 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 *MC*s in $\mathcal{M}$, the upper bound for their sizes is $max(t_e) = t_{max}$.

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 $O(t_{max} \cdot 2^{t_{max}})$. Note that, *EUMC* will be called at most $3^{|V|/3}$ times because G can be divided into at most $3^{|V|/3}$ *MC*s by *BKPivot*. The time complexity of *EMMC* is $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$.

For Function *FRPMC*, first, we need to sort all α-cliques in R. Here, we have counted the number of cliques in some of real datasets and find that there are very few large α-*MC*s, and their sizes are concentrated in a few numbers, as shown in Table 3. Thus, we sort R by counting sort. Line 17 takes $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$ time by counting sort because there are $t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3}$ α-cliques to be sorted.

From Theorem 2, we know that the maximum size of each clique C is bounded by t_{max}. We only update the inverted lists for the vertices in α-*MC*s which means the sizes of L are bounded by k(lines 24 to 30). For each clique C in R, line 21 takes $O(k \cdot t_{max})$ 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 $k \cdot 2^{t_{max}}$ cliques to find all the (k, α)-*MC*s from R.

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

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

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

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 $\mathcal{M}$ to store all the *MC*s computed by Function *BKPivot*. In the second step, we need to maintain all the α-*MC*s computed by Function *EMMC* from each *MC* in $\mathcal{M}$. However, we only need to verify at most $k \cdot 2^{t_{max}}$ α-cliques in the last step. The reason for this problem is that the three steps are performed separately.

Based on this observation, we propose an optimized algorithm *Top-KMC+*, i.e., Algorithm 3, which combines the three steps into one step to compute α-*MC*s. First, we use a priority queue R for maintaining the α-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 *MC*s without considering the probability information by calling Function *BKPivoit*. Different from *Top-KMC*, when an *MC* g is output by Function *BKPivoit*, we immediately compute α-*MC*s form g by Function *ENUMERATE* (lines 2 to 3), such that we do not need to maintain an additional set $\mathcal{M}$ to store all the *MC*s computed by Function *BKPivoit*, where G can be divided into at most $3^{|V|/3}$ *MC*s. From Theorem 3 we know that, we only need to use the result set R to maintain the largest $k \cdot 2^{t_{max}}$ α-cliques. After getting all the $k \cdot 2^{t_{max}}$ α-cliques, we call Function *FRPMC* to find the (k, α)-*MC*s from R(lines 4 to 5).

Function *ENUMERATE* is used to maintain the largest $k \cdot 2^{t_{max}}$ α-cliques computed so far. Specifically, if $|R|$ is less than $k \cdot 2^{t_{max}}$, we directly add the new generated α-clique into R(lines 8 to 10); otherwise, we need to check whether each new generated α-clique C is larger than the smallest clique C' in R. If $size(C) > size(C')$, 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 of $k \cdot 2^{t_{max}} = 5 \cdot 2^4 = 80$ to maintain the α-cliques. Compared to Algorithm 2, which maintains a result set R of $t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3} = 3340$ in the worst case, Algorithm 3 saves large storage space.

When we get the first $MCg_1 = \{v_1, v_2, v_3\}$ from Function *BKPivoit*, Function *ENUMERATE* computes α-*MC*s from g_1. In Function *ENUMERATE*, since the current $|R| < k \cdot 2^{t_{max}} = 5 \cdot 2^4 = 80$, we directly compute α-*MC*s and add each new generated α-*MC* into R, where when $|R| = 80$ we will replace the new generated α-*MCC* with the smallest one C' if $size(C) > size(C')$. Then we get the result set $R = \{\{v_1, v_2, v_3\}\}$. Next, similar to the above process, after computing α-*MC*s from the second *MC* $g_2 = \{v_1, v_3, v_4\}$, we get $R = \{\{v_1, v_2, v_3\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}\}$, where R is a priority queue with counting sort. Similarly, when the last α-*MC* $\{v_7, v_{10}, v_{11}\}$ is computed from the last *MC* $\{v_7, v_{10}, v_{11}\}$, we get the final $R = \{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_7, v_9\}\}$. Finally, we call Function *FRPMC* to verify the result set R to remove the pseudo-α-*MC*s, and output the first k α-*MC*s $\{\{v_5, v_6, v_7, v_8\}, \{v_1, v_2, v_3\}, \{v_7, v_{10}, v_{11}\}, \{v_1, v_4\}, \{v_3, v_4\}\}$.

Algorithm 3: Top-KMC+

Input: $G(V, E, \beta), k, \alpha$
Output: (k, α)-*MC*s

```
R ← ∅   /*R is a priority queue maintaining the result*/
foreach MCg form BKPivot (G) do
    ENUMERATE (R, g, k, α)
FRPMC (R)
output first k α-MCs of R
Function ENUMERATE(R, g, k, α)
if |R| < k · 2^tmax then
    foreach α-clique C form EMMC (g, α, R) do
        if |R| < k · 2^tmax then
            enQueue(R, C)
        else
            REPLACE (R, C)
else
    if |g| >|getTail(R)| then
        foreach α-clique C form EMMC (g, α, R) do
            REPLACE (R, C)
    else
        return
Function REPLACE (R, C)
if size(C) >|getTail(R)| then
    deQueue(R)
    enQueue(R, C)
```

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 $3^{|V|/3}$ *MC*s computed by Function *BKPivoit*, which means Function *ENUMERATE* need to be called at most $3^{|V|/3}$ times.

The time cost of Function *ENUMERATE* is dominated by Function *EMMC* and counting sort on R. The time complexity of Function *EMMC* is $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$, which we have analysed in the previous section. Since R is bounded by $k \cdot 2^{t_{max}}$, the time complexity of counting sort on R is $k \cdot 2^{t_{max}}$. Note that, compared to the counting sort in Algorithm 2, whose time complexity is $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$, the sort in Algorithm 3 is more efficient. Since $O(t_{max} \cdot 3^{|V|/3}) \gg O(k)$, the time cost of Function *ENUMERATE* is $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$.

The difference between Function *FRPMC* in Algorithm 3 and Algorithm 2 is that we do not need to execute line 17 of Function *FRPMC* in Algorithm 2, because the priority queue R has been sorted in Function *ENUMERATE*. Thus, the time complexity of Function *FRPMC* in Algorithm 3 is $O(k^2 \cdot t_{max} \cdot 2^{t_{max}})$, where we have analysed in the previous section.

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

VII. EXPERIMENTS

We conducted extensive experiments to evaluate the performance of our proposed algorithms. The compared algorithms include: (1) *BASIC* algorithm (the basic algorithm described in Section IV); (2) *Top-KMC* algorithm(described in Section V); (3) *Top-KMC+* algorithm(described in Section VI). All algorithms were implemented in C++ and compiled by Microsoft Visual Studio 2019. All the experiments were conducted on the operating system of Windows 10, 64 bits, with an Intel Core i5 1.90GHz CPU and 8GB DDR3-RAM.

A. DATASETS

We used 12 datasets to evaluate the performance of the three algorithms. The datasets include: anthra[1], mtbrv[1], agrocyc[1], ecoo[1], vchocyc[1], amaze[2], kegg[3], xmark[4], nasa[4], citeseer[5], go[6], yago[7]. Among them, anthra, mtbrv, 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 $|V|$, $|E|$, $\overline{d}$ and t_{max} represent the number of vertices, the number of edges, the average of degrees and the maximum truss number in the graph.

[1] ecocyc.org
[2] www.amaze.ulb.ac.be
[3] www.genome.ad.jp/kegg
[4] docs.basex.org/wiki/Statistics
[5] citeseer.ist.psu.edu
[6] www.geneontology.org
[7] mpi-inf.mpg.de/yago-naga/yago

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	5,605	6,537	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
mtbrv	9,602	10,245	2.1	4
citeseer	10,720	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

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

Dataset	*BASIC*	*Top-KMC*	*Top-KMC+*
amaze	1,956	89	74
kegg	2,208	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	873	730

B. COMPARISON OF RUNNING TIME

Table 5 to Table 8 show the comparison of running time. We first vary α by fixing $k = 5$, then vary α by fixing $k = 50$, such that to observe the impacts of both α 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 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, mtbrv, 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 $O(t_{max} \cdot 2^{t_{max}} \cdot 3^{|V|/3})$, which 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. By comparing the results, it can be concluded that our algorithms are more effective especially on sparse graphs. Additionally, *Top-KMC+* is faster than *Top-KMC* on all datasets. The reasons are twofolds. First, *Top-KMC+* only maintian a result set of $k \cdot 2^{t_{max}}$ 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.

1) THE IMPACT OF α

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

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	1,549	1,501
anthra	17,888	800	787
ecoo	18,556	941	848
agrocyc	18,166	878	758

TABLE 7. Running time(*ms*), when $\alpha = 0.1$, $k = 50$.

Dataset	*BASIC*	*Top-KMC*	*Top-KMC+*
amaze	1,956	91	76
kegg	2,208	114	98
nasa	4,087	300	264
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

TABLE 8. Running time(*ms*), when $\alpha = 0.5$, $k = 50$.

Dataset	*BASIC*	*Top-KMC*	*Top-KMC+*
amaze	1,634	94	77
kegg	2,019	122	117
nasa	4,140	310	265
xmark	4,724	321	260
yago	6,730	1,034	979
go	5,656	412	388
vchocyc	10,224	510	468
mtbrv	10,833	521	444
citeseer	13,324	1,622	1,498
anthra	17,888	809	712
ecoo	18,556	965	886
agrocyc	18,166	884	767

varying α from 0.1 to 0.9 fixing $k = 100$ on 12 uncertain datasets. According to the experimental results, we can see that with the increase of α, the running time of *Top-KMC+* is increasing. Because in the Function *EMMC*, we first check whether the probabilities of the *MC*s are no less than the threshold α. The larger the α, the less *MC*s satisfy the condition, which means that more *MC*s need to be further processed by Function *EUMC*.

2) THE IMPACT OF k

We only report the figures 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 fixing $\alpha = 0.5$, 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 α-*MC* until we get the

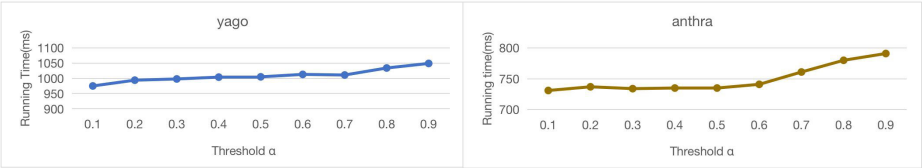


FIGURE 2. Running time of *Top-KMC+*, when $k = 100$ varying α.

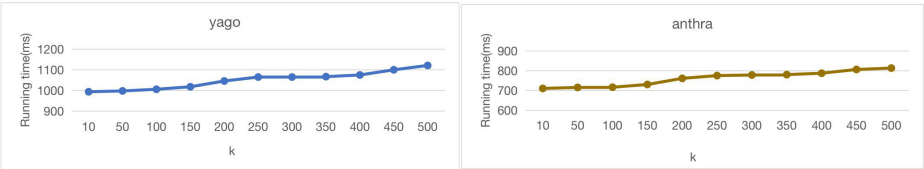


FIGURE 3. Running time of *Top-KMC+*, when $\alpha = 0.5$ varying k.

TABLE 9. The number of cliques each algorithm need to store, when $\alpha = 0.1, k = 50$.

	BASIC		*Top-KMC*			*Top-KMC+*	
Datasets	$\|R\|$	|V|	$\|\mathcal{M}\|$	$\|R\|$	|V|	$k \cdot 2^{t_{max}}$	|V|
amaze	3,394	6,900	3,270	3,394	13,641	400	912
kegg	3,680	7,492	3,558	3,680	14,788	400	932
nasa	6,377	12,836	6,264	6,377	25,506	400	882
xmark	7,025	14,050	7,025	7,025	28,100	200	400
yago	34,957	90,577	37,357	34,957	199,207	6,400	12,853
go	12,333	25,284	11,790	12,333	49,905	400	1,200
vchocyc	9,893	20,019	9,764	9,893	39,935	400	1,033
mtbrv	9,995	20,229	9,893	9,995	40,371	800	1,839
citeseer	38,558	97,944	29,258	38,558	184,472	25,600	72,028
anthra	12,875	25,948	12,739	12,875	51,782	400	998
ecoo	13,073	26,392	12,927	13,073	52,670	400	1,046
agrocyc	13,101	26,480	12,984	13,101	52,863	400	1,078

largest α-*MC*s. 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 α-*MC*s, *Top-KMC* needs to store all the *MC*s and α-*MC*s, *Top-KMC+* only needs to store first $k \cdot 2^{t_{max}}$ α-*MC*s to find (k, α)-*MC*s. In Table 9, we counted the number of α-*MC*s in the result set R of *BASIC*, the number of *MC*s in $\mathcal{M}$ and the number of α-*MC*s in R of *Top-KMC*, the number of α-*MC*s in R, $k \cdot 2^{t_{max}}$, 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+* 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.

VIII. CONCLUSION

This paper presents two efficient algorithms for enumerating *top-K* α-*MC*s over uncertain graphs, namely *Top-KMC* and *Top-KMC+*. Our algorithms are different from existing algorithms, where existing algorithms rank α-*MC*s according to their probabilities while our algorithms return k α-*MC*s with the most number of vertices satisfying that their probabilities $\geq \alpha$. *Top-KMC* works in three steps for α-*MC*s 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 amaze, kegg, xmark, vchocyc, 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 citeseer), *Top-KMC* and *Top-KMC+* are on average about 10 times faster than *BASIC*. For storage space, the space *Top-KMC+* is average about 17 times less than *BASIC*, and 33 times less than *Top-KMC*.

REFERENCES

[1] C. C. Aggarwal, *Managing and Mining Uncertain Data* (Advances in Database Systems), vol. 35. Norwell, MA, USA: Kluwer, 2009.

[2] E. Adar and C. Re, "Managing uncertainty in social networks," *IEEE Data Eng. Bull.*, vol. 30, no. 2, pp. 15–22, Jul. 2007.

[3] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 58, no. 7, pp. 1019–1031, 2007.

[4] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proc. 12th Int. Conf. Inf. Knowl. Manage. (CIKM)*, New Orleans, LA, USA, Nov. 2003, pp. 556–559.

[5] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa, "Injecting uncertainty in graphs for identity obfuscation," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1376–1387, Jul. 2012.

[6] F. N. Abu-Khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova, "On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput," in *Proc. Comput. Biol., Int. Conf. Res. Trends Sci. Technol.*, 2005.

[7] I. Koch, T. Lengauer, and E. Wanke, "An algorithm for finding maximal common subtopologies in a set of protein structures," *J. Comput. Biol.*, vol. 3, no. 2, pp. 289–306, Jan. 1996.

[8] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs," in *Proc. Res. Comput. Mol. Biol., 14th Annu. Int. Conf. (RECOMB)*, Lisbon, Portugal, Apr. 2010, pp. 456–472.

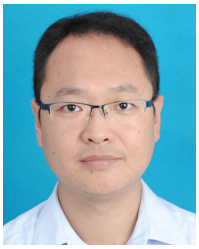
[9] U. Kuter and J. Golbeck, "Using probabilistic confidence models for trust inference in web-based social networks," *ACM Trans. Internet Technol.*, vol. 10, no. 2, pp. 1–23, May 2010.
[10] Y. Mehmood, F. Bonchi, and D. García-Soriano, "Spheres of influence for more effective viral marketing," in *Proc. Int. Conf. Manage. Data*, San Francisco, CA, USA, Jun. 2016, pp. 711–726.
[11] Y. T. H. Kawahigashi, "Modeling ad hoc sensor networks using random graph theory," in *Proc. 2nd IEEE Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2005, pp. 104–109.
[12] B. Yang, D. Wen, L. Qin, Y. Zhang, L. Chang, and R.-H. Li, "Index-based optimal algorithm for computing K-cores in large uncertain graphs," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Macao, China, Apr. 2019, pp. 64–75.
[13] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proc. 31st Int. Conf. Very Large Data Bases*, Trondheim, Norway, Aug. 2005, pp. 721–732. 2005.
[14] L. Qin, R.-H. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Sydney, NSW, Australia, Aug. 2015, pp. 965–974.
[15] N. Modani and K. Dey, "Large maximal cliques enumeration in large sparse graphs," in *Proc. 15th Int. Conf. Manage. Data* (International School of Information Management), S. Chawla, K. Karlapalem, and V. Pudi, Eds. Mysore, India: Computer Society of India, Dec. 2009.
[16] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *Algorithm Theory* (Lecture Notes in Computer Science), vol. 3111, T. Hagerup and J. Katajainen, Eds. Humlebaek, Denmark: Springer, Jul. 2004, pp. 260–272.
[17] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Seoul, South Korea, Apr. 2015, pp. 243–254.
[18] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding top-K maximal cliques in an uncertain graph," in *Proc. IEEE 26th Int. Conf. Data Eng. (ICDE)*, Long Beach, CA, USA, Mar. 2010, pp. 649–652.
[19] R.-H. Li, Q. Dai, G. Wang, Z. Ming, L. Qin, and J. X. Yu, "Improved algorithms for maximal clique search in uncertain networks," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Macao, China, Apr. 2019, pp. 1178–1189.
[20] A. Rashid, M. Kamran, and Z. Halim, "A top down approach to enumerate α-maximal cliques in uncertain graphs," *J. Intell. Fuzzy Syst.*, vol. 36, no. 4, pp. 3129–3141, Apr. 2019.
[21] R. Márquez and R. Weber, "Overlapping community detection in static and dynamic social networks," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Melbourne, VIC, Australia, Jan. 2019, pp. 822–823.
[22] *Proceeding of the Second International Multi-Symposium of Computer and Computational Sciences*, IEEE Comput. Soc., Univ. Iowa, Iowa City, IA, USA, Aug. 2007.
[23] J. Meena and V. S. Devi, "Overlapping community detection in social network using disjoint community detection," in *Proc. IEEE Symp. Comput. Intell.*, Cape Town, South Africa, Dec. 2015, pp. 764–771.
[24] L. Bai, X. Cheng, J. Liang, and Y. Guo, "Fast graph clustering with a new description model for community detection," *Inf. Sci.*, vols. 388–389, pp. 37–47, May 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, Apr. 2014.
[26] M. E. J. Newman, "The structure of scientific collaboration networks," *Proc. Nat. Acad. Sci. USA*, vol. 98, no. 2, pp. 404–409, 2001.
[27] N. Alduaiji, A. Datta, and J. Li, "Influence propagation model for clique-based community detection in social networks," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 2, pp. 563–575, Jun. 2018.
[28] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by H*-graph," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Indianapolis, IND, USA, Jun. 2010, pp. 447–458.
[29] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Paris, France, Jun. 2009, pp. 199–208.
[30] K. Pathak, S. Mane, and J. Srivastava, "Who thinks who knows who? Socio-cognitive analysis of email networks," in *Proc. 6th IEEE Int. Conf. Data Mining*, Hong Kong, Dec. 2006, pp. 466–477.
[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 *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 895–904.
[32] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-K clique search," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Seoul, South Korea, Apr. 2015, pp. 387–398.
[33] S.-V. Sanei-Mehri, A. Das, and S. Tirthapura, "Enumerating top-K quasi-cliques," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Seattle, WA, USA, Dec. 2018, pp. 1107–1112.
[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, Apr. 2020, Art. no. 104867.
[35] Z.-N. Zou and R. Zhu, "Mining top-K maximal cliques from large uncertain graphs," *Chin. J. Comput.*, vol. 36, no. 10, pp. 2146–2155, Mar. 2014.
[36] Y. Che, Z. Lai, S. Sun, Y. Wang, and Q. Luo, "Accelerating truss decomposition on heterogeneous processors," *Proc. VLDB Endowment*, 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, p. 3, Dec. 2013.

JING BAI was born in Harbin, Heilongjiang, China, in 1989. She received the B.S. degree in electronic information engineering from Northeast Forestry University, in 2012, and the M.S. degree in electronic and communication engineering from the East China Normal University, Shanghai, China, in 2015. She is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Donghua University, Shanghai.

Her research interests include query processing and optimization on graph data.

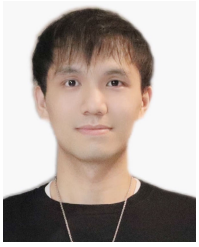
JUNFENG ZHOU was born in Xi'an, Shanxi, China, in 1977. He received the B.S. and M.S. degrees in computer science from Yanshan University, in 1999 and 2002, respectively, and the Ph.D. degree in computer applications from the Renmin University of China, Beijing, China, in 2009.

From 2009 to 2017, he was a Professor with 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.



MING DU was born in Hulin, Heilongjiang, China, in 1975. He received the B.S. degree in electronic technology from Chinese Textile University, in 1998, and the M.S. degree in control theory and control engineering and the Ph.D. degree in management science and engineering from Donghua University, Shanghai, China, in 2005 and 2013, respectively.

Since 2010, he has been an Associate Professor with Donghua University. His research interests include information retrieval techniques, data analysis and mining, and natural language processing.



PENG ZHONG was born in Wuhan, Hubei, China, in 1997. He received the B.S. degree in automation engineering from Wuhan Polytechnic University, in 2018, and the M.S. degree in computer science from Donghua University, Shanghai, China, in 2020.

His research interests include query processing and optimization on graph data.