2019 IEEE 35th International Conference on Data Engineering (ICDE)

Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks

Jiafeng Hu[†] Reynold Cheng[†] Kevin Chen-Chuan Chang[‡] Aravind Sankar[‡] Yixiang Fang[†*] Brian Y.H. Lam[§]
[†]Department of Computer Science, The University of Hong Kong, Hong Kong SAR, China
[‡]Department of Computer Science, University of Illinois at Urbana-Champaign, US
[§]Metabolic Research Laboratories, University of Cambridge, UK
[†]{jhu, ckcheng, yxfang}@cs.hku.hk; [‡]{kcchang, asankar3}@illinois.edu; [§] yhbl2@cam.ac.uk

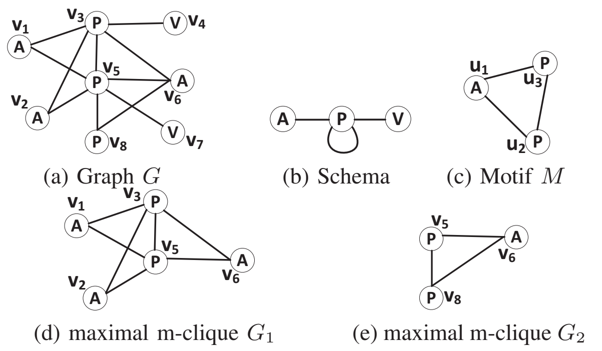
***Abstract*—We study the discovery of cliques (or "complete" subgraphs) in heterogeneous information networks (HINs). Existing clique-finding solutions often ignore the rich semantics of HINs. We propose *motif clique*, or *m-clique*, which redefines subgraph completeness with respect to a given *motif*. A motif, essentially a small subgraph pattern, is a fundamental building block of an HIN. The m-clique concept is general and allows us to analyse "complete" subgraphs in an HIN with respect to desired high-order connection patterns. We further investigate the maximal m-clique enumeration problem (MMCE), which finds all *maximal m-cliques* not contained in any other m-cliques. Because MMCE is NP-hard, developing an accurate and efficient solution for MMCE is not straightforward. We thus present the `META` algorithm, which employs advanced pruning strategies to effectively reduce the search space. We also design fast techniques to avoid generating duplicated maximal m-clique instances. Our extensive experiments on large real and synthetic HINs show that `META` is highly effective and efficient.**

I. Introduction

Heterogeneous information networks (HINs), such as bibliographical databases, co-purchasing graphs, and biological networks, have received a lot of interest from research and industry communities [20], [24], [30]. Nodes of HINs are associated with *labels*, allowing them to capture more sophisticated or "high-order" semantics than unlabelled graphs. In Fig. 1a, for example, a bibliographical graph G consists of three kinds of nodes, namely *Author* (A), *Paper* (P), and *Venue* (V). The rich information contained in an HIN enables important analysis tasks, including similarity search [33], [36], clustering [37], and classification [20].

Cliques. In this paper, we study the problem of finding *cliques* from an HIN. By definition, a clique is a complete subgraph, i.e., all nodes in the clique are adjacent to each other. Thus, a clique contains nodes that are closely related (e.g., a clique in a social network can reveal users who are close friends). A maximal clique is a clique that is not a subgraph of any larger clique. For example, in G (Fig. 1a), one of its maximal cliques is G_2 (Fig. 1e). The problem of discovering all the maximal cliques, called *maximal clique enumeration* (MCE), has been well studied (e.g., [1], [9], [17], [39]). Cliques have been extensively used in social community detection [16], hierarchy detection through email networks [10], financial network analysis [4], and co-expressed gene group detection [29].

∗ Yixiang Fang is the corresponding author.

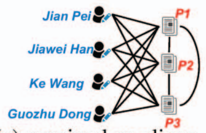


(a) Graph G (b) Schema (c) Motif M (d) maximal m-clique G_1 (e) maximal m-clique G_2

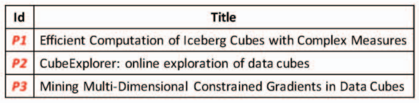
Fig. 1: Illustrating m-cliques for a bibliographical network (A: Author, P: Paper, V: Venue).

Cliques for HINs. Our main goal is to investigate the notion of a clique for an HIN, which contains labelled nodes. Why is this an issue? Let us consider the bibliographical network G in Fig. 1a again. Suppose that we want to find out co-authors who exhibit a close collaboration relationship. Can we get this information from a clique of G? Unfortunately, this is not possible, because an HIN is often associated with a schema [36], which serves as a template for a graph, and tells how many types of nodes there are in the graph and where the possible edges exist. Fig. 1b shows the schema of G, which depicts the connections allowed between nodes with labels "A", "P", and "V". Because no edges between any two "A" nodes are allowed, and a clique is a complete subgraph, there does not exist any clique that has two or more "A" nodes. Hence, we cannot find any clique that contains two or more co-authors from Fig. 1a. Traditional cliques (e.g., Fig. 1e) simply cannot capture the relationships among two or more co-authors. We believe that the sense of a clique in an HIN is still important. In the above example, we may wish to find the "clique" of co-authors who collaborate on every paper. However, the notion of "complete subgraph" needs to be changed, because not every pair of nodes can be connected in an HIN (e.g., an author cannot be linked to another author). How should we rethink the notion of cliques for HINs, whose nodes are labelled and edges are structured by schemas?

Motifs. To address the above question, we incorporate *motifs*



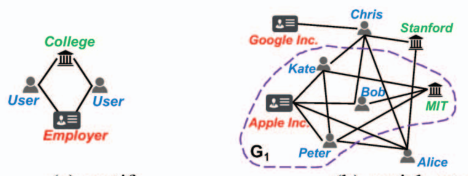
(a) maximal m-clique



Id	Title
P1	Efficient Computation of Iceberg Cubes with Complex Measures
P2	CubeExplorer: online exploration of data cubes
P3	Mining Multi-Dimensional Constrained Gradients in Data Cubes

(b) Papers

Fig. 2: A maximal m-clique for DBLP.



(a) motif (b) social network

Fig. 3: A maximal m-clique for a social network.

to the clique definition. To understand this, let us review the background of motifs. *Motif-based analysis* has emerged as an important tool for HIN insight discovery. A motif, also known as *higher-order structure* or *graphlet*, is a small subgraph pattern. As pointed out by [27], [31], a motif is a fundamental building block of large and complex networks, and it enables "high-order semantics" analysis for HINs. Fig. 1c illustrates a triangle motif M, which describes: (1) an author writes at least two papers; (2) a paper is cited by another one. Motifs are useful in a wide range of graph problems, such as motif discovery [14], [26], graph clustering [2], [41], community search [18], and motif frequency estimation [19], [28], [35].

m-Cliques. We now present the motif-clique (or *m-clique* in short). Intuitively, an m-clique generalizes a clique; it is a "complete subgraph" in terms of a user's defined patterns (motifs), rather than edges. An m-clique is therefore a "higher-order" clique based on a user-given motif, capturing the desired relationship among node labels. More specifically, given an HIN G and a motif M, an m-clique is an induced subgraph G' of G, such that for every node set H of G' containing the same number of nodes with M and sharing the same set of node labels with M, M is isomorphic to the subgraph of G' induced by H. Figs. 1d and 1e show two m-cliques of G (Fig. 1a), based on the motif M shown in Fig. 1c. Both G_1 and G_2 are induced subgraphs of G. More importantly, M is isomorphic to every 3-node induced subgraph of G_1 with labels {"A", "P", "P"} (e.g., $\{v_1, v_3, v_5\}$ and $\{v_2, v_3, v_5\}$); M is also isomorphic to the three nodes of G_2. From G_1, we can see that all the three co-authors (i.e., v_1, v_2, and v_6) wrote the two related papers (i.e., v_3 and v_5). Therefore, G_1 reveals the identities of close collaborators, as well as their publications. Notice that this cannot be obtained by using a traditional clique. In this paper, we focus on *maximal m-cliques*, i.e., m-cliques which are not included in a larger m-clique. As shown in Fig. 1, both G_1 and G_2 are maximal m-cliques. Next, we illustrate m-cliques with two more examples on real datasets.

Scenario 1: DBLP. Fig. 2 shows a maximal m-clique G' based on the motif M (Fig. 1c) obtained from the DBLP dataset. This subgraph contains four authors and three papers with titles shown in Fig. 2b. Notice that M is subgraph isomorphic to every 3-node induced subgraph of G' with labels {"A", "P", "P"} (e.g., the subgraph induced by {"Jiawei Han", "P1", "P2"}). Thus, G' reflects that these four authors have a close collaboration on papers related to data cubes. Observe that this m-clique cannot be obtained by traditional clique solutions, because there is no edge between any two authors in the schema of DBLP. By finding out all the maximal m-cliques in DBLP with the motif M, we can obtain different groups of research collaborators and their related publications.

Scenario 2: Social networks. Consider a social network (e.g., Facebook or LinkedIn), which contains both friendship relations and user attributes. As shown in Fig. 3b, this HIN contains 3 kinds of nodes: "User", "Employer", and "College". Suppose that a headhunting company intends to find people who studied in the same college and worked/are working for the same company. This task cannot be accomplished by a traditional clique, because the underlying schema does not allow an edge between an employer and a college node. Let us see how this can be achieved by an m-clique, with a motif shown in Fig. 3a. This motif depicts that we want to find a cohesive community, with at least two users associated with the same college and employer. The resulting maximal m-clique G_1, shown in Fig. 3b, indicates that Bob, Kate, and Peter are MIT alumni, and they also worked for the Apple company.

Challenges and solutions. Given a large HIN G and a motif M, we aim to discover all the maximal m-cliques of G efficiently. We term this problem *maximal m-clique enumeration*, or MMCE. As pointed out by [23], the MCE problem (for finding maximal cliques) is NP-hard. Because MMCE is a generalization of MCE, finding all the maximal m-cliques from G is extremely costly, especially when G is large. Moreover, to our understanding, there are no previous attempts for solving the MMCE problem.

To tackle MMCE, we borrow the insight from existing MCE solutions ([1], [9], [17], [39]), which generally follow a *node expansion* process: given a candidate subgraph for a clique, grow it by including a new node. Our proposed solution also follows this paradigm. However, as m-cliques are more complex, adopting MCE algorithms requires a significant amount of effort. We propose the `META` algorithm, which iteratively performs subgraph isomorphism and maximal m-clique detection. By introducing the dominance relationship between nodes, we propose principled strategies to facilitate node expansion. We also design early-stop pruning criteria to reduce the search space needed. To avoid computing duplicated maximal m-cliques during node expansion, we utilize a set-trie structure to perform duplication checking.

We have performed an extensive evaluation of our algorithms on large real and synthetic HINs. Three case studies on bibliographical databases, co-purchasing graphs, and biological networks demonstrate the applicability of m-cliques on a wide range of applications. Furthermore, `META` is highly scalable, and is several orders of magnitude faster than baseline solutions. For a 10 million-node HIN, our best algorithm only needs five seconds to return 1,000 maximal m-cliques.

Organization. We review related work in Section II. Section III formulates the MMCE problem. Section IV reviews an algorithm for MCE, which forms the basis of our solutions. In Section V, we present the basic solution META-Basic. We study advanced pruning strategies for MMCE in Sections VI, VII and VIII, respectively. Section IX reports experimental results. We conclude in Section X.

II. Related Work

Maximal clique enumeration (MCE), which finds maximal cliques (or maximal subgraphs whose nodes are adjacent to each other) from unlabelled graphs, has been extensively studied [1], [9], [17], [39]. Researchers have also examined bi-cliques (for bi-partite graphs) [42] and k-partite cliques (for k-partite graphs) [25]. In fact, these cliques are special cases of the m-clique (Section III). In this paper, we propose motif-based cliques for HINs, which consider both structure and label information of HINs.

Because MCE is NP-hard [23], researchers have developed pruning strategies to reduce the search space and time costs [6], [8], [17]. Most of these approaches were based on the classical BK algorithm [6], which uses backtracking to explore the search space effectively. Our META algorithm is also inspired by BK. However, adapting BK to solve MMCE is not trivial. This is because we have to incorporate motifs in the m-clique discovery process, and this makes the solution more sophisticated. We will discuss the main challenges of extending BK in detail in Section IV.

Motifs, or small patterns, are fundamental building blocks for large networks [27], [31]. It has been studied in various domains, such as neuroscience [34], biology [40], and social networks [5], [31]. Recently, several important motif-related problems have been examined. For instance, Gurukar et al. [14] studied the mining of communication motifs from dynamic interaction networks. They developed COMMIT, a technique that converts a dynamic network into a database of sequences, in order to discover communication motifs. In [2], [21], [41], the problem of motif-aware (or higher-order) graph clustering was addressed. Motif conductance, a generalization of the conductance metric for motifs, is employed in the graph clustering process. In [19], [28], [35], the problem of counting or estimating the frequency of motifs was addressed. We are not aware of any work that incorporates motifs in the clique detection problem. We propose the m-clique, and design strategies to retrieve them efficiently from a large HIN.

III. The Maximal m-clique Enumeration Problem

We model a *Heterogeneous Information Network (HIN)* as an undirected and labelled graph $G = (V_G, E_G, L_G, \Sigma_G)$ where V_G is the set of nodes, $E_G \subseteq V_G \times V_G$ is the set of edges, Σ_G is the set of labels, and L_G is a labelling function that assigns each node $v \in V_G$ a single label in Σ_G (denoted by $L_G(v)$)[1]. We use $G[U] = (U, E[U], L_G, \Sigma_G)$ to denote the subgraph of G induced by node set $U \subseteq V_G$, where $E[U] = \{(u,v)|u, v \in U, (u,v) \in E_G\}$. Let $\mathcal{N}_G(v)$ be the set of neighbors of $v \in V_G$ in G. For ease of presentation, we use "graph" to refer to an HIN. As illustrated in Section I, the schema of an HIN expresses all allowable links between node labels (e.g., Fig. 1b). Further, we define a motif $M = (V_M, E_M, L_M, \Sigma_M)$ to be any small connected HIN. Given a graph G, a valid motif for G should follow G's schema, i.e., 1) only contains node labels defined by the schema; and 2) only contains edges between node labels that are allowed by the schema. Typically, M is given by users (e.g., a "triangle" in Fig. 1c). The particular motif used is application specific. We have illustrated the use of motifs in bibliographic and social networks (Sec I), and we will also discuss a few more motifs used in the case studies (Sec IX).

[1]In this paper, we focus on undirected HINs for simplicity. Our techniques can be readily extended to handle edge-labeled and directed HINs.

In this paper, we generalize the definition of cliques for motif-based analysis on HINs. Before describing our m-clique model, let us review the definition of *subgraph isomorphism*.

Definition 3.1 (Subgraph Isomorphism [7]): A motif M is *subgraph isomorphic* to a graph G if there exists an injective mapping ϕ: $V_M \rightarrow V_G$, s.t., $\forall u \in V_M, L_M(u) = L_G(\phi(u))$ and $\forall u, v \in V_M$, if $(u,v) \in E_M$, then $(\phi(u), \phi(v)) \in E_G$, where $\phi(u)$ is the node to which u is mapped.

Clearly, M is subgraph isomorphic to G **iff** M is isomorphic to a subgraph (not necessarily induced) of G. We call an injective mapping from nodes in M to nodes in G a *subgraph isomorphic embedding* of M in G.

Next, we propose a preliminary concept called "label-matched node set". Intuitively, it is a set of nodes such that these nodes have the same labels with the given motif. For example, for the graph G in Fig. 1a and the motif M in Fig. 1c, every 3-node set with labels {"A", "P", "P"}, e.g., $\{v_1, v_3, v_5\}$ and $\{v_6, v_3, v_5\}$, is a label-matched node set of M in G.

Definition 3.2 (Label-matched Node Set): Given a graph G and a motif M, a node set $H \subseteq V_G$ is label-matched with M if H and M have the same number of nodes (i.e., $|H| = |V_M|$), and $\exists$ bijection $\phi : H \rightarrow V_M$, such that $\forall u \in H$, $L_G(u) = L_M(\phi(u))$. We call H a label-matched node set of M in G.

We now present formal definitions of m-clique and maximal m-clique.

Definition 3.3 (Motif Clique): Given a graph G and a motif M, a motif clique (*m-clique* for short) of M in G is an induced subgraph G' of G, such that G' and M have the same set of labels, at least one label-matched node set of M is contained in G' and for each label-matched node set H in G', M is subgraph isomorphic to $G'[H]$ (the induced subgraph).

Definition 3.4 (Maximal m-clique): An m-clique is a maximal m-clique if it is not contained in any other m-clique.

Problem Statement. Given a graph G and a motif M, we study the problem of *maximal m-clique enumeration* (MMCE) which efficiently extracts all maximal m-cliques of M in G.

Example 3.1: For the graph G in Fig. 1a and the motif M in Fig. 1c, there are two maximal m-cliques, as shown in Fig.s 1d (G_1) and 1e (G_2). For G_1, it contains three label-matched node sets: $\{\{v_1, v_3, v_5\}, \{v_2, v_3, v_5\}, \{v_6, v_3, v_5\}\}$. We can observe that M is subgraph isomorphic to all subgraphs of G_1 induced by these label-matched sets. For G_2, V_{G_2} is a label-matched node set and M is subgraph isomorphic to G_2.

TABLE I: Notations

Notation	Description
M and G	Motif and data graph
V_G and E_G	Node set and edge set of G
V_M and E_M	Node set and edge set of M
$G[U]$	Subgraph of G induced by node set U
$L_g(v)$	Label of v in g
$\mathcal{N}_g(v)$	Neighbors of v in g
$\|U\|$	Number of nodes in node set U
$\mathcal{D}_v^U$	Nodes in U dominated by v
n	Number of nodes in G

Generalization. MMCE is a generalization of the well-studied maximal clique enumeration (MCE) problem defined for unlabelled graphs and the maximal k-partite clique problem defined for k-partite graphs. Here, we summarize how MMCE generalizes them in detail:

• When G is an unlabelled graph, the MCE problem [1] searches for all maximal cliques. This is equivalent to MMCE for G, by setting M to be a single edge connecting two unlabelled nodes.

• When G is a k-partite graph ($k \geq 2$), the maximal k-partite clique problem [25] asks for all maximal k-partite cliques of G (i.e., maximal complete k-partite subgraphs of G). Particularly, when $k = 2$, it is the maximal bi-clique problem [42]. This is the same as MMCE for G, where M is a complete graph containing k nodes with k different labels.

Neither MCE nor the maximal k-partite clique problem are designed for general labelled graphs (or HINs). Moreover, there are cliques that can be found by MMCE but not by them. For instance, the maximal m-clique in Fig. 1d is neither a classical clique nor a k-partite clique.

As discussed in [23], MCE is NP-hard. Because MMCE is a generalization of MCE, MMCE is also NP-hard. Nevertheless, we have developed new solutions for MMCE, which run reasonably fast on large HINs in our experiments. We will study these algorithms in the next few sections. We will use "embedding" and "subgraph isomorphic embedding", as well as "label-matched set" and "label-matched node set", in an interchangeable manner. We will simplify $L_g(v)$ and $\mathcal{N}_g(v)$ as $L(v)$ and $\mathcal{N}(v)$, respectively, when the context is clear. The frequently used notations are summarized in Table I.

IV. An Overview of BK

The classical BK algorithm [6], discussed in Section II, is a well-studied and efficient solution for MCE. Our algorithm follows the same paradigm of BK. Hence, before describing our solution, let us review BK briefly. BK is a recursive solution that searches for all maximal cliques in a given unlabelled graph. The intuition is to expand an existing clique to a larger one iteratively. Three *disjoint* sets are maintained in BK: a set U of nodes in the current detected clique, a set $\mathcal{C}$ of candidates for clique expansion (i.e., $\forall v \in \mathcal{C}$, $G[U \cup v]$ is a clique), and a set NOT of forbidden nodes (i.e., $\forall v \in$ NOT, $G[U \cup v]$ is a clique, and all maximal cliques containing $U \cup v$ have already been found). Note that NOT is used to ensure every maximal clique will only be reported once. Algorithm 1 is the pseudocode of BK [39]. Given a graph G, by invoking

Algorithm 1: BK(U, $\mathcal{C}$, NOT)

```
1  if $\mathcal{C} = \emptyset$ then
2      if NOT $= \emptyset$ then
3          Output $G[U]$ as a maximal clique;
4      return;
5  while $\mathcal{C} \neq \emptyset$ do
6      if $\exists v \in$ NOT$, \mathcal{N}(v) \supseteq \mathcal{C}$ then
7          return;
8      sample uniformly at random a node $u \in \mathcal{C}$;
9      $\mathcal{C} \leftarrow \mathcal{C} \backslash u$;
10     $\mathcal{C}_{\texttt{new}} \leftarrow \mathcal{C} \cap \mathcal{N}(u)$;
11     NOT$_{\texttt{new}} \leftarrow$ NOT $\cap \mathcal{N}(u)$;
12     BK($U \cup u, \mathcal{C}_{\texttt{new}}$, NOT$_{\texttt{new}}$);
13     NOT $\leftarrow$ NOT $\cup u$;
```

BK($\emptyset, V_G, \emptyset$), we can find all maximal cliques in G. The following are 4 key points.

• **Initial state.** Initially, both U and NOT are empty and all nodes are considered as candidates (i.e., $\mathcal{C} = V_G$).

• **Node expansion.** The algorithm proceeds by iteratively adding each candidate u from $\mathcal{C}$ to U, updating $\mathcal{C}$ and NOT to include only neighbors of u (lines 10-11), and making a recursive call to find all maximal cliques containing $u \cup U$ (line 12). Then, we add u to NOT to exclude it from consideration in future cliques (line 13).

• **Recursion termination.** Through recursive invocations, a maximal clique is reported when both of the sets $\mathcal{C}$ and NOT are empty. The status that $\mathcal{C} = \emptyset$ and NOT $\neq \emptyset$, means $G[U]$ is a clique, but it is not a maximal clique since each node in NOT can be added into $G[U]$ to become a larger clique. Hence, we can backtrack directly without reporting.

• **Early stop.** Note that we can stop to further expand the current clique if there exists a node v in NOT which is adjacent to all nodes in $\mathcal{C}$ (lines 6-7). Because the node v will be kept in NOT (i.e., NOT $\neq \emptyset$) even when $\mathcal{C} = \emptyset$. We call this operation "*early stop pruning*".

A running example of BK is provided in the Appendix section.

To solve the MMCE problem, we propose a framework which follows the BK algorithm to recursively enumerate maximal m-cliques. However, adapting BK to solve MMCE is not trivial due to the followings 4 major challenges:

Challenge 1. The initial state of adopting BK for MMCE is not clear. It is not efficient to find maximal m-cliques by expanding from an empty set which is used in BK, since any maximal m-clique must contain at least one embedding of the given motif. We need to seek for a new framework which can leverage existing solutions for subgraph isomorphism.

Challenge 2. In node expansion, for the BK algorithm, after picking a new node u, we only need to check whether nodes in $\mathcal{C}$ (resp. NOT) is adjacent to u for computing $\mathcal{C}_{\texttt{new}}$ (resp. NOT$_{\texttt{new}}$). This step costs $\mathcal{O}(n)$. However, the problem of examining whether a node can be added into the current m-clique to form a larger one, is challenging. As we will present in Section VI, it is NP-hard. We need to derive novel pruning strategies to improve the efficiency.

Challenge 3. The early stop pruning in BK is not suitable for MMCE since the definition of m-clique is much more

Algorithm 2: `META-Basic`(G, M)

```
Input: Graph $G$ and motif $M$
Output: All maximal m-cliques of $M$ in $G$
1  for each embedding $S$ found by an existing subgraph isomorphism
     algorithm do
2      $S' \leftarrow \{u | u \in V_G \backslash S, \mathcal{N}(u) \cap S \neq \emptyset\}$;
3      $\mathcal{C} \leftarrow$ Refine($S, S'$);
4      GetMMC($S, \mathcal{C}, \emptyset$);
5  procedure Refine($U, X$)
6      $X_{\texttt{new}} \leftarrow \emptyset$;
7      for each $x \in X$ do
8          if mCliqueCheck($U, x$) is true then
9              $X_{\texttt{new}} \leftarrow X_{\texttt{new}} \cup x$;
10     return $X_{\texttt{new}}$;
```

Algorithm 3: `mCliqueCheck`(U, x)

```
1  for each label-matched set $H$ of $M$ in $G[U \cup x]$ containing $x$ do
2      if $G[H]$ is not an embedding of $M$ then
3          return $false$;
4  return $true$;
```

complicated (general) than traditional clique. It is extremely expensive to check whether a node in `NOT` can be kept even when $\mathcal{C} = \emptyset$.

We address Challenge 1 by proposing a novel framework for MMCE which considers embeddings of the given motif as initial states. However, based on that framework, a new problem we encounter is the computation of duplicated maximal m-cliques since a maximal m-clique can be found starting from different embeddings. This poses a new challenge (**Challenge 4**) of avoiding duplication which is not faced by the BK algorithm. We utilize a set-trie structure to avoid computing duplicate maximal m-cliques.

V. META: The Maximal m-clique Enumeration Algorithm

We now present the **M**aximal m-clique **E**numera**T**ion **A**lgorithm (`META` for short), which aims to solve the MMCE problem quickly. In this section, we discuss its basic version (called `META-Basic`). In the following sections, we will study how to further enhance the performance of `META-Basic`.

Given an m-clique $G[U]$, how to check whether it is a maximal one? Intuitively, $G[U]$ is maximal if $\forall U' \subseteq (V_G \backslash U)$ ($U' \neq \emptyset$), $G[U \cup U']$ is not an m-clique. However, it is expensive to check the maximality in this way because of the huge number of U'. In the following, we introduce two preliminary lemmas to simplify such checking.

Lemma 5.1: Given a graph G, a motif M, and two m-cliques $G[U_1]$ and $G[U_2]$ with node set $U_1 \subseteq U_2$, for any node set $U_3 \subseteq (U_2 \backslash U_1)$, $G[U_1 \cup U_3]$ is an m-clique.

We put all proofs of the lemmas of this paper into the Appendix section.

Lemma 5.2 (Maximality): Given a graph G and a motif M, an m-clique $G[U]$ is maximal if and only if $\forall v \in V_G \backslash U$, $G[U \cup v]$ is not an m-clique.

By the definition of m-clique, we can observe that any m-clique must contain at least one embedding of M. Thus, according to Lemma 5.2, we can claim that every maximal m-clique can be found by expanding from any embedding it contains. Based on this observation, we develop `META-Basic` (Algorithm 2), which (1) enumerates all embeddings of M in G, and (2) for each embedding S, finds out all maximal m-cliques containing S.

Specifically, we first use a state-of-the-art subgraph isomorphism algorithm (e.g., VF3 [7]) to find all embeddings of M sequentially. Once a new embedding S is found, we compute the initial candidate set $\mathcal{C}$ such that $\forall u \in \mathcal{C}$, $G[S \cup u]$ is an m-clique of M. A straightforward way is to examine, for every node $u \in V_G \backslash S$, whether $G[S \cup u]$ is an m-clique. We observe a simple pruning trick which is to check only those nodes which are adjacent to some node in S, i.e., $\{u | u \in V_G \backslash S, \mathcal{N}(u) \cap S \neq \emptyset\}$. Because if a node u is not adjacent to S, for any label-matched set $H \subseteq (S \cup u)$ containing u, $G[H]$ is disconnected which means it is not an embedding. Here we encapsulate the candidate computation into the `Refine` procedure. Given an m-clique $G[U]$ and a set X of nodes, `Refine`(U, X) returns a set $X_{\texttt{new}}$ such that $\forall x \in X_{\texttt{new}}$, $G[U \cup x]$ is an m-clique. Furthermore, in `Refine`, the `mCliqueCheck` procedure (Algorithm 3) is to check whether $G[U \cup x]$ is an m-clique. Intuitively, by the definition of m-clique, $G[U \cup x]$ is an m-clique if and only if each label-matched set H in $G[U \cup x]$ is an embedding of M. Hence, `mCliqueCheck` returns $true$ if all label-matched sets are embeddings; otherwise, it returns $false$.

After obtaining $\mathcal{C}$, we invoke `GetMMC` to find all maximal m-cliques containing the embedding S. Before introducing `GetMMC` in detail, let us give an example of executing `META-Basic`.

Example 5.1: Consider the running example in Fig. 1. We first compute embeddings of M sequentially. There are 4 embeddings, namely $S_1 = \{v_1, v_3, v_5\}$, $S_2 = \{v_2, v_3, v_5\}$, $S_3 = \{v_6, v_3, v_5\}$ and $S_4 = \{v_6, v_5, v_8\}$. For each embedding, we detect all maximal m-cliques containing it. Take S_1 as an example. We first get all nodes in $V_G \backslash S_1$ which are adjacent to any node S_1, i.e., $S' = \{v_2, v_4, v_6, v_7, v_8\}$. Next, we invoke `Refine`$(S_1, S')$ to compute $\mathcal{C}$. In particular, for each node $u \in S'$, we test whether $G[S_1 \cup u]$ is an m-clique. Neither nodes v_4 and v_7 can be added into the result set since M does not contain the label "V", nor can node v_8 since there exists a label-matched set $\{v_1, v_5, v_8\}$ in $G[S \cup v_8]$ which is not an embedding of M. After invoking `Refine`, we get the candidate set $\mathcal{C} = \{v_2, v_6\}$. Next, `GetMMC`$(S_1, \mathcal{C}, \emptyset)$ is invoked to find out all maximal m-cliques containing S_1. Finally, G_1 in Fig. 1d is found and reported. After processing all these 4 embeddings, maximal m-cliques G_1 and G_2 are obtained, as shown respectively in Fig.s 1d and 1e. Notice that in this example, G_1 can be detected through the expansion of S_1, S_2, and S_3, respectively. Hence, duplicated maximal m-cliques are detected. We will discuss how to avoid it in Section VIII.

A. The GetMMC Algorithm

In what follows, we illustrate `GetMMC` in detail. The purpose of `GetMMC` in Algorithm 2 (line 4) is to find out all maximal

Algorithm 4: GetMMC(U, $\mathcal{C}$, NOT)

```
1  if $\mathcal{C} = \emptyset$ then
2      if NOT $= \emptyset$ then
3          Output $G[U]$ as a maximal m-clique;
4      return;
5  while $\mathcal{C} \neq \emptyset$ do
6      sample uniformly at random a node $u \in \mathcal{C}$;
7      $\mathcal{C} \leftarrow \mathcal{C} \backslash u$;
8      $\mathcal{C}_{new} \leftarrow$ Refine($U \cup u, \mathcal{C}$);
9      NOT$_{new} \leftarrow$ Refine($U \cup u$, NOT);
10     GetMMC($U \cup u, \mathcal{C}_{new}$, NOT$_{new}$);
11     NOT $\leftarrow$ NOT $\cup\, u$;
```

m-cliques containing the given embedding S.

As shown in Algorithm 4, similar to the BK algorithm, we maintain three *disjoint* sets of nodes: (1) the set U of nodes in the current search path ($G[U]$ is an m-clique), (2) the set $\mathcal{C}$ of nodes that can be added into U, i.e. $\forall v \in \mathcal{C}$, $G[U \cup v]$ is an m-clique, and (3) the set NOT of nodes where for each node $v \in$ NOT, $G[U \cup v]$ is an m-clique and all maximal m-cliques containing $U \cup v$ have been found. The main differences between BK and GetMMC are listed as follows. First, for the initial state, BK is $(\emptyset, V_G, \emptyset)$ while the initial state of GetMMC is $(S, \mathcal{C}, \emptyset)$. Second, in the node expansion step, when computing $\mathcal{C}_{new}$ and NOT$_{new}$ after the selection of node $u \in \mathcal{C}$, in BK we can compute them by simply checking whether a node is adjacent to u. However, in GetMMC we invoke Refine to filter out those nodes that cannot be added into $G[U \cup u]$ which is expensive. Third, the early stop pruning in BK no longer applies in GetMMC. Next, we show the correctness of GetMMC.

Lemma 5.3: [Correctness of Algorithm 4] Given a graph G, a motif M, an embedding S of M in G and the initial candidate set $\mathcal{C}$, GetMMC$(S, \mathcal{C}, \emptyset)$ can find out all maximal m-cliques containing S without duplication.

Based on Lemma 5.3 and the fact any maximal m-clique must contain at least one embedding of M, we can claim that META-Basic(G, M) is correct, i.e., it can find out all maximal m-cliques of M in G.

Complexity. Given a graph G and a motif M, let $\rho = \mathcal{O}(\binom{n}{|V_M|})$ denote the time cost of finding all embeddings of M in G, and α denote the time cost of mCliqueCheck. In addition, the worst case time complexity of BK without early stop pruning is $\mathcal{O}(n!)$. Hence, the worst case time cost of the META-Basic algorithm is $\mathcal{O}(\rho + \alpha * n!)$. Because during the node expansion, META-Basic costs α to check whether a node can be kept while it only costs $\mathcal{O}(1)$ in BK (i.e., adjacent check). We consider finding all embeddings (subgraph isomorphism) as a black box, any state-of-the-art algorithms can be used to speed up that part.

Discussions. Up to now, we have illustrated our META-Basic algorithm. Below are several aspects which we focus on to improve the efficiency of META-Basic.

• As discussed in Challenge 2 (in Section IV), the problem of examining whether a node can be added into the current m-clique to form a larger one (the mCliqueCheck algorithm), is NP-hard (as we will prove later in Section VI). We present an efficient pruning method to improve the performance by introducing the dominance relationship between nodes.

• Different with BK, we do not use any early stop checking technique in the current GetMMC algorithm due to high complexity of deciding whether a node from NOT can be kept even when $\mathcal{C} = \emptyset$. In Section VII, we derive an early stop pruning strategy customized for m-cliques to check whether we can stop to further expand the current m-clique early.

• Another issue of META-Basic is that duplicated maximal m-cliques can be reported. Since we detect maximal m-cliques for each embedding individually, a maximal m-clique which contains multiple embeddings will be found multiple times. The size of search space can be reduced significantly if we can avoid to re-compute those maximal m-cliques in an early stage. We will discuss how to avoid duplications in Section VIII.

VI. Advanced Node Expansion

In this section, we focus on improving the performance of node expansion, i.e., checking whether a node can be incorporated into an m-clique to become a larger one. First, we show that this problem is NP-hard. Then, we propose a pruning strategy to speed up node expansion.

The following lemma shows that it is NP-complete to decide whether an m-clique still is an m-clique after removing an arbitrary edge.

Lemma 6.1: Given a motif M, an m-clique $g = (V_g, E_g)$ with $|V_g| = |V_M|$, and a graph $g' = (V_g, E_g \backslash e)$ where $e \in E_g$, the problem of deciding whether g' is an m-clique is NP-complete.

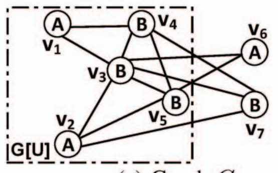
Based on Lemma 6.1, we can have the following Lemma.

Lemma 6.2: Given a motif M, an m-clique $g = (V_g, E_g)$, a node $u \notin V_g$, let g' be a super graph of g where $V_{g'} = V_g \cup u$ and there is no constraint on edges from u to nodes in V_g. The problem of deciding whether g' is an m-clique is NP-hard.

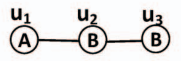
Although it is NP-hard to identify new candidates, we have proposed a pruning strategy to improve the efficiency in some cases. The intuition of the proposed strategy is to utilize: 1) the fact that each label-matched set in the current m-clique is an embedding of M; and 2) the dominance relationship between the new node and those nodes in the current m-clique, to facilitate the checking of whether a label-matched set containing the new node is an embedding. Next, we first show Lemma 6.3 for checking whether a label-matched set is an embedding, and then show Lemma 6.4 for checking whether a node can be incorporated into the current m-clique.

Lemma 6.3: Given a motif M, an m-clique $G[U]$ and a node $u \in V_G \backslash U$, for any label-matched set $H \subseteq (U \cup u)$ containing u, if $\exists v \in U \backslash H$, s.t., $L(v) = L(u)$ and $(\mathcal{N}(u) \cap H) \supseteq (\mathcal{N}(v) \cap (H \backslash u))$, $G[H]$ is an embedding of M.

Example 6.1: For the graph G and motif M in Fig. 4 with node-labels being {"A", "B"}, suppose we have found an m-clique $G[U]$ and would like to check whether the label-matched set $H = \{v_6, v_3, v_5\}$ containing $u = v_6$ is an embedding. Since there exists a node v_1, such that $L(v_1) = L(v_6)$="A" and $(\mathcal{N}(v_6) \cap H) = \{v_3, v_5\} \supseteq (\mathcal{N}(v_1) \cap (H \backslash v_6)) = \{v_3\}$, by Lemma 6.3 we can claim that $G[H]$ is an embedding of M without any isomorphism checking.



(a) Graph G



(b) Motif M

Fig. 4: An example of pruning in candidate identification.

The above pruning can speed up the checking for a single label-matched set. However, to check whether a node u can be incorporated into the current m-clique $G[U]$, we still need to check label-matched sets one by one until a label-matched set which is not an embedding is found. If $G[U \cup u]$ is an embedding, we need to check all label-matched sets. Since the number of such sets is exponential, the computational cost is extremely high. Next, we will discuss on how to reduce the number of label-matched sets that need to be checked significantly. Let us first introduce the concept of *dominance*.

Definition 6.1 (dominance): Given a graph G, an induced subgraph $G[U]$ and two nodes $u \in V_G \backslash U$ and $v \in U$ with $L(u) = L(v)$, v is dominated by u if $(\mathcal{N}(v) \cap U) \subseteq (\mathcal{N}(u) \cap U)$. Let $\mathcal{D}_u^U$ denote the set of nodes in U dominated by u.

As shown in Fig. 4, we have $\mathcal{D}_{v_6}^U = \{v_2\}$ and $\mathcal{D}_{v_7}^U = \{v_5\}$. Given an m-clique $G[U]$ and a node $u \in V_G \backslash U$, for any label-matched set $H \subseteq U \cup u$ containing u, if $\mathcal{D}_u^U \nsubseteq H$, $G[H]$ is an embedding by Lemma 6.3. Let t be the number of nodes in M with label being $L(u)$. For any H, we have $|H \cap \mathcal{D}_u^U| \leq t$. Hence, in some cases, $\mathcal{D}_u^U \nsubseteq H$ is always true. We can derive the following lemma based on the aforementioned observations.

Lemma 6.4: Given a motif M, an m-clique $G[U]$, a node $u \in V_G \backslash U$, the dominance set $\mathcal{D}_u^U$, and the number t of nodes in M with label being $L(u)$, $G[u \cup U]$ is an m-clique **iff**:

1) $1 + |\mathcal{D}_u^U| > t$; otherwise
2) for each label-matched set $H \subseteq (U \cup u)$ containing $u \cup \mathcal{D}_u^U$, $G[H]$ is an embedding of M.

Example 6.2: For the graph G and motif M in Fig. 4, suppose we have found an m-clique $G[U]$ and would like to check whether $G[v_6 \cup U]$ and $G[v_7 \cup U]$ are m-cliques. We illustrate the checking based on Lemma 6.4 as follows.

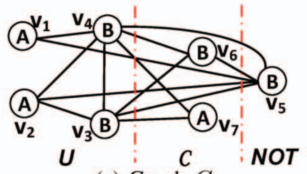
Case 1 ($u = v_6$)**.** We first compute $\mathcal{D}_{v_6}^U = \{v_2\}$ and $t = 1$. Since $1+|\mathcal{D}_{v_6}^U| > t$, $G[v_6 \cup U]$ is an m-clique based on Lemma 6.4 (1), without any isomorphism checking.

Case 2 ($u = v_7$)**.** We have $\mathcal{D}_{v_7}^U = \{v_5\}$ and $t = 2$. Since $1+|\mathcal{D}_{v_7}^U| <= t$, by Lemma 6.4 (2), we only need to check all label-matched set containing $u = v_7$ and $\mathcal{D}_{v_7}^U = \{v_5\}$. Since the subgraph induced by the label-matched set $\{v_2, v_5, v_7\}$ is not an embedding of M, $G[v_7 \cup U]$ is not an m-clique.

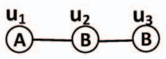
Based on the aforementioned lemmas, we propose an advanced algorithm, called `mCliqueCheckPlus` for node expansion to replace its basic version `mCliqueCheck`. As shown in Algorithm 5, we first compute $\mathcal{D}_u^U$ and t. If $1 + |\mathcal{D}_u^U| > t$, the algorithm returns $true$ directly; otherwise, we test each label-matched set H in $G[u \cup U]$ containing $u \cup \mathcal{D}_u^U$ one by one. Particularly, we invoke a subgraph isomorphism algorithm

Algorithm 5: `mCliqueCheckPlus`(U, u)

```
1 $\mathcal{D}_u^U \leftarrow \{v | v \in U, L(v) = L(u), (\mathcal{N}(v) \cap U) \subseteq (\mathcal{N}(u) \cap U)\}$;
2 $t \leftarrow$ the number of nodes in $M$ with label being $L(u)$;
3 if $1 + |\mathcal{D}_u^U| > t$ then
4     return $true$;
5 else
6     for each label-matched set $H$ in $G[u \cup U]$ containing $(u \cup \mathcal{D}_u^U)$ do
7         if $G[H]$ is not an embedding of $M$ then
8             return $false$;
9     return $true$;
```



(a) Graph G



(b) Motif M

Fig. 5: A counter-example of trivially adapting the early stop pruning of BK to `GetMMC`.

to check whether $G[H]$ is an embedding. Once we find that there exists a label-matched set which is not an embedding, the algorithm returns $false$ immediately. Otherwise, it returns $true$ since for all such H, $G[H]$ is an embedding. The time complexity of computing $\mathcal{D}_u^U$ is $\mathcal{O}(|U| * |\mathcal{N}(u)|) = \mathcal{O}(n^2)$. Even though in the worst case, we still need to check all label-matched sets in $G[u \cup U]$, it works well in practice as we will show in the evaluation section.

VII. Early stop pruning

In this section, we focus on proposing a method to check whether `GetMMC` can stop expanding early for the current invocation. As show in Algorithm 4 (`GetMMC`), at the beginning of the while loop, we select a node u from $\mathcal{C}$ without checking whether there exists a node from `NOT` can be kept even when $\mathcal{C} = \emptyset$ (so called early stop pruning in BK) due to the high complexity. Given the sets U, $\mathcal{C}$ and `NOT` in `GetMMC`, by analogy with the early stop pruning of BK (checking whether $\exists v \in$ `NOT`$, \mathcal{N}(v) \supseteq \mathcal{C}$), a simple adaptation is to check whether $\exists v \in$ `NOT` such that $\forall u \in \mathcal{C}$, $G[U \cup v \cup u]$ is an m-clique. However, even though there exists such a node v, we cannot ensure that v will be kept even when $\mathcal{C} = \emptyset$ since there may exist some set $\mathcal{C}' \subseteq \mathcal{C}$ such that $G[U \cup \mathcal{C}']$ is an m-clique while $G[U \cup \mathcal{C}' \cup v]$ is not an m-clique. Following is an example.

Example 7.1: For the graph G and motif M in Fig. 5, let $U = \{v_1, v_2, v_3, v_4\}$, $\mathcal{C} = \{v_6, v_7\}$ and `NOT` $= \{v_5\}$. Although both $G[U \cup v_6 \cup v_5]$ and $G[U \cup v_7 \cup v_5]$ are m-cliques, we cannot stop to expand U, since $G[U \cup v_6 \cup v_7]$ is a maximal m-clique which has not been reported. In other words, v_5 will be removed from `NOT` after adding both v_6 and v_7 to U.

The above example shows that the early stop pruning of BK cannot be adapted to solve the MMCE problem. Next, let us go deeper by introducing a lemma about the sufficient condition of early stop when searching maximal m-cliques.

Algorithm 6: `EarlyStopCheck`$(U, \mathcal{C}, \texttt{NOT})$

```
1 for each $v \in \texttt{NOT}$ do
2     compute $\mathcal{D}_v^{U \cup \mathcal{C}}$;
3     $\mathcal{D}' \leftarrow \mathcal{D}_v^{U \cup \mathcal{C}} \backslash \mathcal{C}$;
4     $t \leftarrow$ the number of nodes in $M$ with label being $L(v)$;
5     if $1 + |\mathcal{D}'| > t$ then
6         return $true$;
7 return $false$;
```

Lemma 7.1: When invoking `GetMMC`$(U, \mathcal{C}, \texttt{NOT})$, it will not output anything if $\exists v \in \texttt{NOT}, \forall \mathcal{C}' \subseteq \mathcal{C}$,

$$G[U \cup \mathcal{C}'] \text{is an m-clique} \Rightarrow G[U \cup \mathcal{C}' \cup v] \text{is an m-clique.}$$

Since the number of subsets of $\mathcal{C}$ is $2^{|\mathcal{C}|}$, it costs $\mathcal{O}(2^{|\mathcal{C}|} * \alpha)$ time to check whether all subsets satisfy the condition mentioned in Lemma 7.1, where α is the time cost of checking whether a subgraph is an m-clique (it is NP-hard as analyzed in Section VI). Thus, it is too expensive to do this. *Can we find a way to do early stop pruning efficiently?* We observe that the dominance relationship introduced in Section VI can also be used here. The following lemma shows that in some cases, we can stop to expand remaining nodes in $\mathcal{C}$.

Lemma 7.2: Given an m-clique $G[U]$, a set $\mathcal{C}$ of nodes and a node $v \notin (U \cup \mathcal{C})$, let $\mathcal{D}_v^{U \cup \mathcal{C}}$ be the set of nodes in $U \cup \mathcal{C}$ dominated by v. Let $\mathcal{D}' = \mathcal{D}_v^{U \cup \mathcal{C}} \backslash \mathcal{C}$ and t be the number of nodes in M with label being $L(v)$. If $1 + |\mathcal{D}'| > t$, we have,

$$\forall \mathcal{C}' \subseteq \mathcal{C}, G[U \cup \mathcal{C}'] \text{is an m-clique} \Rightarrow G[U \cup \mathcal{C}' \cup v] \text{is an m-clique.}$$

Notice that in Lemma 7.2 we use $\mathcal{D}'$ rather than $\mathcal{D}_v^{U \cup \mathcal{C}}$ directly. Because we need to ensure that for any $\mathcal{C}' \subseteq \mathcal{C}$, if $G[U \cup \mathcal{C}']$ is an m-clique, $G[U \cup \mathcal{C}' \cup v]$ is an m-clique. Hence any node in $\mathcal{C}$ should not be included into the dominance set. As analyzed in Section VI, the dominance set $\mathcal{D}_v^{U \cup \mathcal{C}}$ can be computed in $\mathcal{O}(|U \cup \mathcal{C}| * |\mathcal{N}(v)|)$ time. Hence, we can check whether $1 + |\mathcal{D}'|$ is larger than t efficiently. Based on Lemma 7.2, we devise the `EarlyStopCheck` algorithm as shown in Algorithm 6. Essentially, we iteratively check all nodes in `NOT`. The algorithm returns $true$ if any such node is detected. `EarlyStopCheck` can be invoked at the beginning of the while loop in `GetMMC`. Specifically, we add the following code before line 6 in `GetMMC` (Algorithm 4):

```
if EarlyStopCheck$(U, \mathcal{C}, \texttt{NOT})$ is $true$ then return;
```

Once `EarlyStopCheck` returns $true$, we can stop searching maximal m-cliques containing U since there must exist a node $v \in \texttt{NOT}$ which will be kept even when $\mathcal{C} = \emptyset$.

Interestingly, for the MCE problem which is a special case of MMCE, i.e., G is an unlabelled graph and M is an edge connecting two unlabelled nodes, it is easy to check that `EarlyStopCheck` will return $true$ **iff** $\exists v \in \texttt{NOT}$ with $\mathcal{N}(v) \supseteq \mathcal{C}$. It is consistent with the early stop pruning in BK. Hence, the proposed early stop pruning for m-clique is also a generalization of the one in BK.

VIII. Duplication Avoidance

As shown in Lemma 5.3, for each embedding S, the `GetMMC` algorithm can find all maximal m-cliques of M in G containing S. However, since a maximal m-clique may contain several different embeddings, it can be found independently from different embeddings. For instance, in the example shown in Fig. 1, both embeddings $\{v_1, v_3, v_5\}$ and $\{v_2, v_3, v_5\}$ can lead to the detection of the maximal m-clique G_1 (Fig. 1d).

A naive way to avoid reporting duplicated maximal clique is to store all reported results and check whether it has already existed through sequential scanning. If the answer is yes, it will not be reported. Otherwise, it will be added into the result set. However, there are several drawbacks listed as follows: 1) since the number of maximal m-cliques can be very large, it is not efficient to check whether a new one has already existed in the result set sequentially; 2) it can only avoid reporting those duplicated maximal m-cliques at the final stage, i.e., the computation cost of finding those results cannot be reduced. It would be much better if we can detect early and avoid re-computing those maximal m-cliques which have already been reported in previous invocations of `GetMMC`.

In this section, an efficient algorithm is proposed to avoid computing duplicated maximal m-cliques as early as possible. In particular, we model our problem of avoiding duplicated maximal m-clique generation as the subset query problem. Let us first introduce the subset query problem.

Definition 8.1 (The Subset Query Problem): Let Ψ be a set of ordered symbols, $\mathbb{S}$ be a group of subsets of Ψ, and U be a subset of Ψ. A subset query answers whether there exists an element $S \in \mathbb{S}$ such that $S \subseteq U$.

Let Ψ be the set of node ids (ordered) of G, $\mathbb{S}$ be the set of subgraph embeddings have been examined so far, i.e., all maximal m-cliques containing them have been generated, and U be the current explored m-clique in `GetMMC`. We need not continue extending U for computing maximal m-cliques if $\exists S \in \mathbb{S}$, $S \subseteq U$. Because all maximal m-cliques containing S have already been generated. We use the set-trie structure which is proposed in [32] for efficient subset query processing.

The set-trie is a tree composed of nodes tagged with indices from 1 to n where n is the size of the alphabet (in our problem, n is the number of nodes in G). Essentially, the set-trie is a trie tree [11] originally designed for efficient string matching among a set of strings. However, the set-trie is designed to support subset queries. The root node of the tree is tagged with "$\{\}$" and its children can be the nodes tagged from 1 to n. A node with tag i can have children tagged with numbers greater than i (suppose we have a global order for all labels and there is no duplicate element in all sets). In addition, each node has a flag ("flag_last") indicating whether it is the last element in a set (the default value is $false$). At the beginning, the tree $\mathcal{T}$ only includes the root node which represents an empty set. Once an ordered set S comes, we update the tree $\mathcal{T}$ by invoking the `Insert` function, as shown in Algorithm 7. Note that different from the set-trie proposed in [32], we also maintain the height of each node for further pruning during the query processing. The height of the root node is set to 0.

Fig. 6 shows an example set-trie. A set is represented by a path from root node to a node with flag_last set to $true$ (red nodes). In our problem, since all embeddings (sets) have the same length, the flag_last is set to $true$ **iff** it is a leaf node.

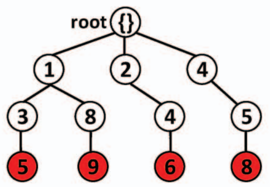


Fig. 6: An example set-trie for sets {1, 3, 5}, {1, 8, 9}, {2, 4, 6} and {4, 5, 8}. Red nodes are the last element in some sets (flag_last=true).

Algorithm 7: Insert($S, \mathcal{T}$)
```
cur_node ← $\mathcal{T}$.root;
for each ordered element $s \in S$ do
    if cur_node.HasChild($s$) is false then
        cur_node.CreateChild($s$, cur_node.height + 1);
    cur_node ← cur_node.GetChild($s$);
cur_node.flag_last ← true;
```

When a new subset query comes, based on the set-trie, the authors of [32] proposed a Depth-First Search (DFS) algorithm to check whether there exists a set $S \subseteq U$. In the following, we introduce a Breath-First Search (BFS) algorithm as well as a pruning strategy utilizing the property that all sets in $\mathcal{T}$ have the same length (e.g., $|V_M|$, the number of nodes in M). Given an ordered set U and a set-trie $\mathcal{T}$, Algorithm 8 returns $true$ if there exists a set $S \subseteq U$; otherwise returns $false$. Specifically, we use $active\_set$ to store all nodes have been visited and may refer to an "end node" of some set. At the beginning, $active\_set$ only contains the root node (line 1). If U is not empty, we examine ordered elements in U one by one. Once $active\_set$ is empty, it returns $false$ (lines 3-4). Otherwise, for each ordered element $u \in U$, we test whether the node in $active\_set$ can be extended (lines 9-13) and whether it can still be kept in the next round (lines 14-15, i.e., the pruning by height). If a node with flag_last being $true$ is visited, the algorithm returns $true$ (lines 11-12). Finally, the algorithm returns $false$ when U is empty.

As analysed in [32], the time cost of SubsetQueryProcess is $\mathcal{O}(c * |\mathcal{T}|)$, where $|\mathcal{T}|$ represents the size of $\mathcal{T}$ and c is a constant which is up to 5 in practice. Next, we illustrate how to avoid detecting duplicated maximal m-cliques by using the set-trie structure. We modify the META-Basic (Algorithm 2) described in Section V as follows.

1) Initialize an empty set-trie $\mathcal{T}$ at the beginning (before line 1);
2) Insert the embedding S into $\mathcal{T}$ after invoking GetMMC($S, \mathcal{C}, \emptyset$).

Algorithm 8: SubsetQueryProcess($U, \mathcal{T}$)
```
1  active_set ← {$\mathcal{T}$.root};
2  while $U \neq \emptyset$ do
3      if active_set = $\emptyset$ then
4          return false;
5      pick the first element $u$ from $U$;
6      $U \leftarrow U \backslash u$;
7      next_active_set ← $\emptyset$;
8      for each element $v \in$ active_set do
9          if $v$.HasChild($u$) is true then
10             $w \leftarrow v$.GetChild($u$);
11             if $w$.flag_last is true then
12                 return true;
13             next_active_set.Add($w$);
14         if $v$.height + $|U| \geq |V_M|$ then
15             next_active_set.Add($v$);
16     active_set ← next_active_set;
17 return false;
```

In addition, we add following codes before line 8 in GetMMC (Algorithm 4):
```
if SubsetQueryProcess($U \cup u, \mathcal{T}$) is true then
    NOT ← NOT $\cup\, u$;
    continue;
```
If SubsetQueryProcess returns $true$, we need not to further expand $U \cup u$ since all maximal m-cliques containing $U \cup u$ have already been found.

So far, we have introduced all optimization strategies for improving META-Basic, namely the advanced node expansion, the early stop pruning and the duplication avoidance technique. We name the algorithm with all these optimizations as META.

IX. Experiments

A. Setup

Datasets. We evaluate the performance of the tested algorithms on both real and synthetic graphs described as follows.

Real graphs. We evaluate the algorithms on five real graphs.

- DBLP: this is an HIN from DBLP [20], which contains 4 labels of nodes: 5,237 papers (P), 5,915 authors (A), 18 venues (V), and 4,479 topics (T). The topics are terms extracted from paper titles. There are 4 types of edges, i.e., A-P, V-P, T-P and P-P. The total number of edges is 51,377 and the average degree is 6.6.
- Amazon[2]: a product co-purchasing network with 1.78M edges, 548K nodes, and 4 labels. Each node has a title and a label that indicates its category (i.e., "Books", "music CDs", "DVDs" and "VHS video tapes"). An edge between products x and y means that x and y appear together in a single purchase.
- Reactome[3]: a publicly available bioinformatics database from the European Bioinformatics Institute (EBI). Reactome is a large, heterogeneous interaction network containing proteins, biochemical reactions, and pathways from multiple biological species [13]. We focus on "Homo Sapiens (human)", which consists of 54,397 nodes, 97,843 edges, with an average degree of 3.6, and 15 distinct labels.
- Yeast [3]: a protein interaction network containing 12,519 edges and 3,112 nodes with an average degree of 8.1, and 71 labels.
- Instacart[4]: a co-purchasing network containing 12,770 edges, 5,240 nodes and 21 labels, where each edge between products x and y means they have been purchased together more than 200 times. Each product (node) has a label that shows its category, e.g., "personal care" and "beverages".

Synthetic graphs. We also test our solutions on graphs generated by an open-sourced benchmark graph generator [22][5]. Because the generator can only produce unlabelled graphs, we associate each node with a label randomly drawn from a set

[2]http://snap.stanford.edu/data/index.html
[3]https://reactome.org/
[4]https://www.instacart.com/datasets/grocery-shopping-2017
[5]http://santo.fortunato.googlepages.com/benchmark.tgz

TABLE II: Statistics of synthetic graphs (**K**=10^3, **M**=10^6).

Param	Description	Values	Default
$size$	#nodes	100**K**, 1**M**, 5**M**, 10**M**	**1M**
$\bar{d}$	average degree	4, 8, 16, 32	**8**
$\|\Sigma\|$	#distinct labels	25, 50, 100, 200	**50**

of distinct labels. Table II summarizes the parameters used to generate synthetic graphs.

Motifs. For case studies, we design the motifs for particular purposes, whereas for efficiency experiments, we randomly create motifs to test with different situations and get the average performance. In particular, a motif is generated as a connected subgraph of the data graph, by conducting random walk on the data graph, as it was done in [3]. For performance evaluation, we create motifs with size in $\{3, 4, 5, 6, 7\}$ (default size is 4), because the size of motifs is bounded from 3 to 7 in real applications [14], [27], [40], [41]. For each data graph, we create 5 motif sets, each of which contains 100 motifs of the same size.

Algorithms. We have tested the following five variants of `META`:

- `META-Basic`: Algorithm 2.
- `META-ES`: Algorithm 2 with **E**arly **S**top pruning only.
- `META-ES-ANE`: Algorithm 2 with both early stop pruning and **A**dvanced **N**ode **E**xpansion included.
- `META-ES-DA`: Algorithm 2 with both early stop pruning and **D**uplication **A**voidance included.
- `META`: Algorithm 2 with all pruning techniques, including advanced node expansion, early stop pruning, and duplication avoidance (i.e., our best algorithm).

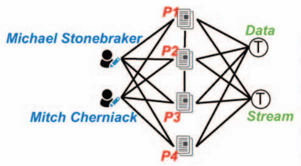
Here we only report the results of one version of `META` (i.e., `META-Basic`) which does not employ early stop pruning. This is because without this functionality, none of our solutions can finish within 24 hours.

Experimental environment. All our algorithms are implemented in C++. For subgraph isomorphism, we use VF3 [7], which is a state-of-the-art subgraph isomorphism algorithm [6]. The experiments are conducted on a 16GB memory machine with Intel(R) Core(TM) i7 CPU@2.3 GHz.

B. Case studies

Scenario 1: collaboration analysis. We perform two studies on DBLP. As discussed in Section I, we have used `META` to find groups of research collaborators (Fig. 2). Here we show another example (Fig. 7), based on the motif in the form of a path connecting three nodes (i.e., "Author-Paper-Topic"). Intuitively, we wish to find all the papers written by authors that share common topics. We discovered 37 maximal m-cliques containing "Michael Stonebraker", each with around 10 nodes and 14 edges on average. The maximal m-clique displayed in Fig. 7 shows that Stonebraker and Cherniack co-authored 4 papers, with two topics ("data" and "stream") in common. Fig. 7b shows the titles of these papers. Note that although there are three groups of nodes in this network, we

[6]http://mivia.unisa.it/datasets/graph-database/vf3-library/



Id	Title
P1	Operator Scheduling in a Data Stream Manager
P2	Aurora: A Data Stream Management System
P3	Load Shedding in a Data Stream Manager
P4	Linear Road: A Stream Data Management Benchmark

(a) a maximal m-clique (b) papers

Fig. 7: Case study 1 (collaboration analysis on DBLP).

cannot run a 3-partite clique query (and obtain the m-clique here). This is because no edge exists between authors and topics in the schema of DBLP.

Scenario 2: purchase analysis. On the Instacart co-purchasing graph, we execute `META` with a 4-node motif shown in Fig. 8b. This motif contains 3 types of grocery items (i.e., "Snacks", "Beverages" and "Breakfast"). The intuition of this motif is to find out those beverages and breakfast products which are frequently bought together with snacks for better product promotion. We discovered 756 maximal m-cliques in 7.9 seconds, each with around 11 nodes and 54 edges on average. Fig. 8a shows a maximal m-clique found by `META`. This subgraph is highly connected, and likely reflects the related items interesting to customers. The result can be used in product promotion (e.g., selling at a discounted price a bundle containing 2 snacks, 1 beverage, and 1 breakfast product extracted from this maximal m-clique).

Scenario 3: biological analysis. We have conducted a case study on the Reactome network. Biologists are interested in analyzing biological complexes or events whose details are not fully known, to help refine reactions/pathways that are not well understood [12]. In this study, we examine whether maximal m-cliques can help on this. We used a 4-node motif (Fig. 9b), which contains 3 types of objects, namely "Complex", "Reaction", and "BlackBoxEvent". Specifically, "Complex" denotes physical entities (i.e. proteins) formed by the association of two or more other entities; "Reaction" refers to bona fide biochemical reactions which have balanced input and output entities; and "BlackBoxEvent" refers to reactions or complex processes where details are not yet established.

Using our `META` algorithm, we discovered 42 maximal m-cliques in 0.2 seconds, each with connected BlackBoxEvents, complexes and reactions. On average, each maximal m-clique contains around 6 nodes and 10 edges. Fig. 9a depicts the largest maximal m-clique discovered which contains 1 BlackBoxEvent, 2 complexes, and 42 reactions. This maximal m-clique allows biologists to understand the BlackBoxEvent ("CD209 activate GTPase RAS") which has mechanisms similar to those of reactions. They share the same inputs or complexes. Maximal m-cliques can thus help biologists to better understand these BlackBoxEvents by integrating previously unconnected reactions and pathways.

C. Efficiency

We now compare the efficiency of `META` with its variants. We vary the number of maximal m-cliques to be reported, from 10^2 to 10^4 with 10^3 being the default, as it was done in [15], [38] for subgraph isomorphism enumeration. We treat

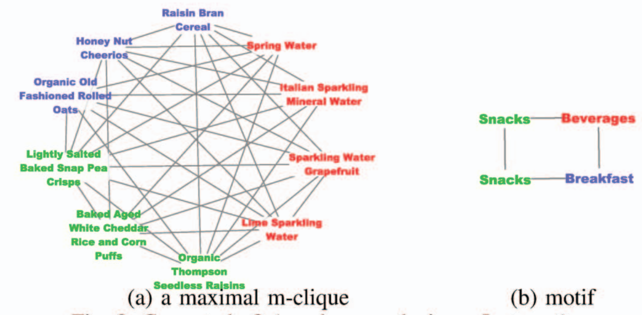
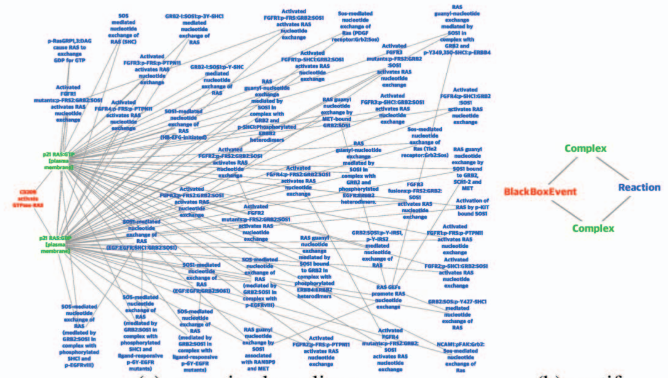

Raisin Bran Cereal
Honey Nut Cheerios
Spring Water
Organic Old Fashioned Rolled Oats
Italian Sparkling Mineral Water
Lightly Salted Baked Snap Pea Crisps
Sparkling Water Grapefruit
Baked Aged White Cheddar Rice and Corn Puffs
Lime Sparkling Water
Organic Thompson Seedless Raisins
Snacks
Beverages
Snacks
Breakfast
(a) a maximal m-clique (b) motif

Fig. 8: Case study 2 (purchase analysis on Instacart).



(a) a maximal m-clique (b) motif

Fig. 9: Case study 3 (biological analysis on Reactome).

the running time of a query as infinite (**Inf**) if the query set cannot finished in 24 hours.

Effect of motif size. We examine the impact of motif sizes on the performance of META. Each set of motifs with the same size contains 100 motif instances. Each query is run 3 times, and we report the average running time of these 100 queries in Fig. 10, on real graphs. We observe the following. First, the running time of META-Basic is "Inf". Second, META-ES (with early stop pruning) can finish within the time limit for most cases, thus showing the importance of early stop pruning. Third, both META-ES-ANE and META-ES-DA perform better than META-ES, because both advanced node expansion and duplication avoidance can further improve the efficiency. Fourth, META, which includes all pruning techniques, performs the best; it is at least 3 orders of magnitude faster than META-Basic. Fifth, the running time of META increases with the motif size in general. Because the cost of checking the maximality of an m-clique increases with the motif size. We conclude that our pruning techniques for node expansion, early stop and duplication avoidance effectively enhance the performance of META.

Effect of the number of maximal m-cliques. Now we evaluate the effect of the number of maximal m-cliques reported on real graphs. We vary the number of maximal m-cliques reported from 10^2 to 10^4. Fig. 11 shows the average running time of the META algorithm on different graphs over different motif sizes. As expected, the processing time increases when more results are reported. Furthermore, the increment of running time is near linear to the increasing number of maximal m-cliques reported in general.

Scalability testing. We test the scalability of META on synthetic graphs by varying $size$ (default 1**M**), $\bar{d}$ (default 8), and $|\Sigma|$ (default 50). As shown in Table III, META is highly scalable on large graphs (e.g., it only costs 4.68 seconds on the 10**M**-node graph). We observe that the processing time of META increases with $size$. The larger the graph, the larger is the search space. Also, the processing time of META increases with the average degree $\bar{d}$. When $\bar{d}$ is larger, the graph has more edges, thereby increasing the running time of META. Finally, the cost of META decreases with the increase of $|\Sigma|$, due to fewer embeddings and maximal m-cliques.

TABLE III: Scalability testing of META (motif size=4, in seconds)

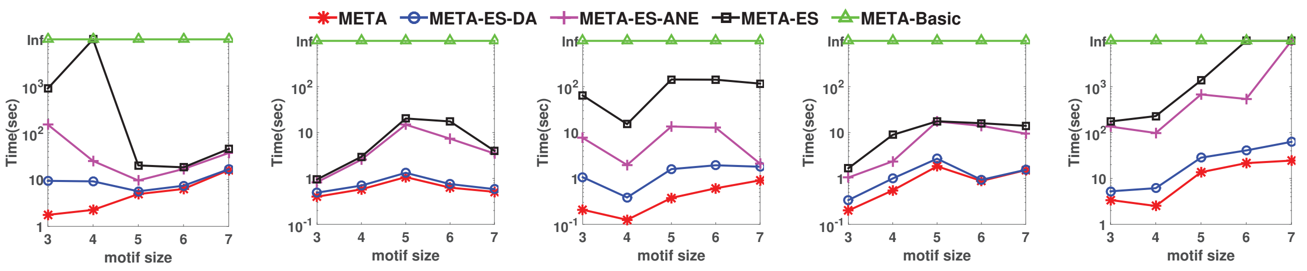
$size$	100**K**	1**M**	5**M**	10**M**
Time	0.05	0.8	2.79	4.68
$\bar{d}$	4	8	16	32
Time	0.39	0.8	0.92	1.01
$\|\Sigma\|$	25	50	100	200
Time	1.05	0.8	0.18	0.16

X. Conclusions

In this paper, we propose the m-clique, which incorporates motifs in a clique for motif-based analysis on HINs. We also formulate the maximal m-clique enumeration (MMCE) problem. To tackle this problem, we propose the META algorithm, which employs novel pruning strategies for node expansion and early stop pruning, as well as the duplication avoidance strategy. Our experiments show that META is highly effective and efficient. In particular, for effectiveness, three case studies in different domains demonstrate the application scenarios of maximal m-cliques. For efficiency, META achieves several orders of magnitude improvement in performance compared with the basic version, which is attributed to our powerful optimization techniques. In the future, we will extend META to handle more rich information on a graph (e.g., nodes with multiple attributes or labels, and edges with directions and labels). We will also study how META can be run in a distributed and parallel computing environment.

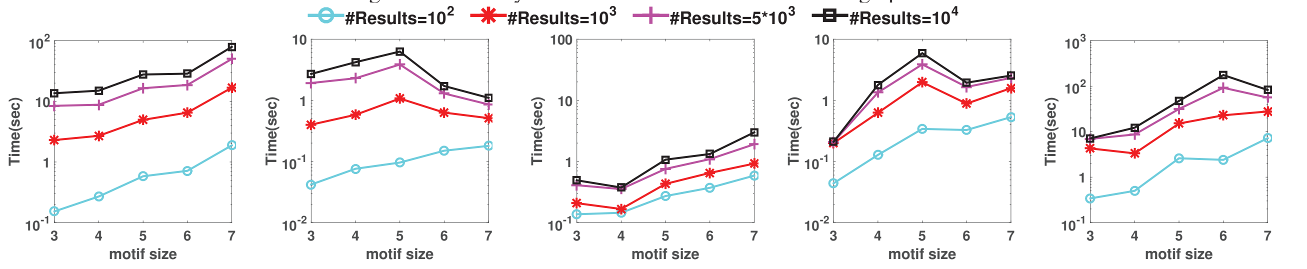
XI. Acknowledgments

We would like to thank the reviewers for the insightful comments. Reynold Cheng, Jiafeng Hu, and Yixiang Fang were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116, 106150091, and 17205115) and the University of Hong Kong (Projects 104004572, 102009508, and 104004129). Kevin C.C. Chang and Aravind Sankar were supported by the National Science Foundation IIS 16-19302 and IIS 16-33755 and Zhejiang University ZJU Research 083650. Brian Lam was supported by the UK Medical Research Council (MRC) Metabolic Disease Unit (MRC_MC_UU_12012/1 & MRC_MC_UU_12012/5), a Wellcome Trust Strategic Award(100574/Z/12/Z), Society for Endocrinology Early Career Grant and the Helmholtz Alliance ICEMED. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.



(a) DBLP (b) Amazon (c) Reactome (d) Yeast (e) Instacart

Fig. 10: Efficiency evaluation over motif size on real graphs.



(a) DBLP (b) Amazon (c) Reactome (d) Yeast (e) Instacart

Fig. 11: Efficiency evaluation over the number of maximal m-cliques reported on real graphs.

References

[1] E. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973.

[2] A. R. Benson et al. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[3] F. Bi et al. Efficient subgraph matching by postponing cartesian products. In *SIGMOD*, pages 1199–1214, 2016.

[4] V. Boginski et al. Statistical analysis of financial networks. *Computational statistics & data analysis*, 48(2):431–443, 2005.

[5] A. Bonato et al. Dimensionality of social networks using motifs and eigenvalues. *PloS one*, 9(9):e106052, 2014.

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

[7] V. Carletti et al. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *TPAMI*, 40(4):804–818, 2018.

[8] J. Cheng et al. Fast algorithms for maximal clique enumeration with limited memory. In *KDD*, pages 1240–1248, 2012.

[9] A. Conte et al. Finding all maximal cliques in very large social networks. In *EDBT*, pages 173–184, 2016.

[10] G. Creamer et al. Segmentation and automated social hierarchy detection through email network analysis. In *WEBKDD*, pages 40–58. 2009.

[11] R. De La Briandais. File searching using variable length keys. In *Western joint computer conference*, pages 295–298, 1959.

[12] A. Fabregat et al. Reactome pathway analysis: a high-performance in-memory approach. *BMC bioinformatics*, 18(1):142, 2017.

[13] A. Fabregat et al. The reactome pathway knowledgebase. *Nucleic acids research*, 46(D1):D649–D655, 2017.

[14] S. Gurukar et al. Commit: A scalable approach to mining communication motifs from dynamic networks. In *SIGMOD*, pages 475–489, 2015.

[15] W.-S. Han et al. Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD*, pages 337–348, 2013.

[16] R. A. Hanneman and M. Riddle. Introduction to social network methods, chapter 11: cliques., 2005.

[17] B. Hou et al. Efficient maximal clique enumeration over graph data. *Data Science and Engineering*, 1(4):219–230, 2016.

[18] X. Huang et al. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[19] M. Jha et al. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, pages 495–505, 2015.

[20] M. Ji et al. Graph regularized transductive classification on heterogeneous information networks. In *ECML-PKDD*, pages 570–586, 2010.

[21] L. Lai et al. Scalable distributed subgraph enumeration. *PVLDB*, 10(3):217–228, 2016.

[22] A. Lancichinetti et al. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

[23] E. L. Lawler et al. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[24] M. Ley. Dblp: some lessons learned. *PVLDB*, 2(2):1493–1500, 2009.

[25] Q. Liu et al. k-partite cliques of protein interactions: A novel subgraph topology for functional coherence analysis on ppi networks. *Journal of theoretical biology*, 340:146–154, 2014.

[26] G. Micale et al. Fast analytical methods for finding significant labeled graph motifs. *DMKD*, 32(2):504–531, 2018.

[27] R. Milo et al. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[28] A. Paranjape et al. Motifs in temporal networks. In *WSDM*, pages 601–610, 2017.

[29] G. A. Pavlopoulos et al. Using graph theory to analyze biological networks. *BioData mining*, 4(1):10, 2011.

[30] N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

[31] N. Pržulj and N. Malod-Dognin. Network analytics in the age of big data. *Science*, 353(6295):123–124, 2016.

[32] I. Savnik. Index data structure for fast subset and superset queries. In *ARES*, pages 134–148, 2013.

[33] C. Shi et al. Relevance search in heterogeneous networks. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 180–191. ACM, 2012.

[34] O. Sporns and R. Kötter. Motifs in brain networks. *PLoS biology*, 2(11):e369, 2004.

[35] L. D. Stefani et al. Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. *TKDD*, 11(4):43, 2017.

[36] Y. Sun et al. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[37] Y. Sun et al. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *TKDD*, 7(3):11, 2013.

[38] Z. Sun et al. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

[39] E. Tomita et al. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[40] S. Wuchty et al. Evolutionary conservation of motif constituents in the yeast protein interaction network. *Nature genetics*, 35(2):176–179, 2003.

[41] H. Yin et al. Local higher-order graph clustering. In *KDD*, pages 555–564, 2017.

[42] Y. Zhang et al. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics*, 15(1):110, 2014.