TED: Towards Discovering Top-k Edge-Diversified Patterns in a Graph Database

ABSTRACT

With an exponentially growing number of graphs from disparate repositories, there is a strong need to analyze a graph database containing an extensive collection of small- or medium-sized data graphs (*e.g.,* chemical compounds). Although subgraph enumeration and subgraph mining have been proposed to bring insights into a graph database by a set of subgraph structures, they often end up with similar or homogenous topologies, which is undesirable in many graph applications. To address this limitation, we propose the *Top-k Edge-Diversified Patterns Discovery problem* to retrieve a set of subgraphs that cover the maximum number of edges in a database. To efficiently process such query, we present a generic and extensible framework called TED which achieves a guaranteed approximation ratio to the optimal result. Three optimization strategies are further developed to improve the performance, and a lightweight version called TEDLITE is designed for even larger graph databases. Experimental studies on real-world datasets demonstrate the superiority of TED to traditional techniques.

1 INTRODUCTION

The graph database that contains a large collection of small- or medium-sized data graphs has become increasingly prevalent in a variety of domains such as biological networks, cheminformatics, drug discovery, and computer vision. Analyzing and mining such a database nurtures a lot of applications, including graph search and classification. While graph analysis varies from application to application, a common phenomenon in these applications is that some subgraph structures (*also known as* graph patterns) play a vital role in characterizing the underlying graph database and building intuitive models for better understanding complex structures. Consequently, subgraph enumeration, which aims to enumerate all subgraphs in a graph database, has been extensively studied in the literature [2–6]. Subgraph enumeration is known to be computationally challenging and memory-consuming since the number of subgraphs in a graph database is exponential to the database size. A complete enumeration and persistence of all subgraphs in a graph database is therefore infeasible. For instance, more than a million compounds and drugs are publicly available from sources such as PUBCHEM [1], AIDS [2] and EMOLECULES [3]. As shown in our experiments in Section 7, subgraph enumeration on only 5,000 graphs in PUBCHEM already uses over 32GB memory.

Frequent subgraph mining alleviates this problem by generating only frequent subgraphs instead of all subgraphs [7–12]. Given a minimum support threshold sup_{min}, a subgraph g is frequent if the fraction of data graph G containing g (*i.e.,* g is a subgraph of G) in a database D is no less than sup_{min}. Intuitively, as the threshold decreases, the number of resulting subgraphs dramatically increases. On the contrary, the number of subgraphs decreases as the threshold increases, making the resulting subgraphs topologically similar to each other with common substructures and no longer representational. In other words, the results lack diversity whose importance has been advocated in the literature [14–17].

[1] https://pubchem.ncbi.nlm.nih.gov/
[2] https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data
[3] https://www.emolecules.com/

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

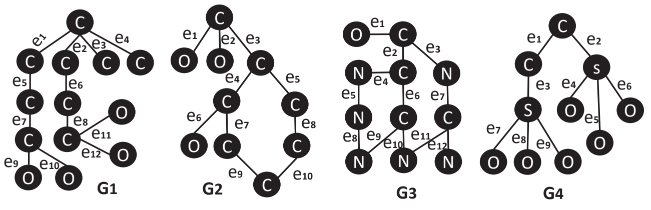
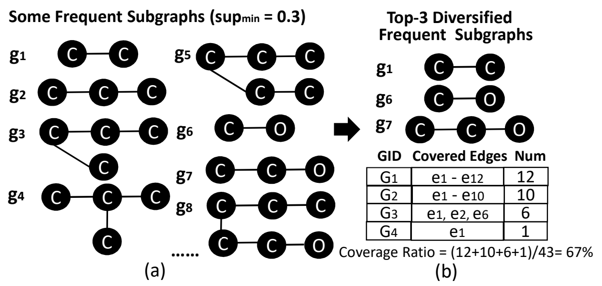


Figure 1: A sample graph database.

EXAMPLE 1. *To illustrate this, consider a graph database containing G_1 to G_4 in Figure 1. Let minimum support $sup_{min} = 0.3$. The database contains many frequent subgraphs, some of which are shown in Figure 2(a). All these subgraphs except g_6 share a common subgraph g_1, which is homogenous and thus redundant. A possible way to alleviate this problem is to select a limited number (*e.g., *k) of diversified ones from all frequent subgraphs, such that the maximum number of edges in the database can be covered. For example, if we select the top-3 diversified frequent subgraphs (*i.e., *g_1, g_6 and g_7) shown in Figure 2(b), they will cover a maximum number of edges (*i.e., *67% edges) in all data graphs from G_1 to G_4. As a comparison, if we just randomly select 3 subgraphs, for example g_3, g_4 and g_5, users cannot obtain any information on graphs G_3 and G_4 since they contain none of the subgraphs.* ■

This example motivates the need for retrieving top-k diversified subgraphs that cover a maximum number of edges in the graph database. Note that not all such subgraphs are frequent, since those frequent subgraphs are often clustered and thus cannot reach a wider edge coverage. For example, g_7 does not cover additional edges that are not covered by g_1 and g_6 (see Figure 2(b)). However, if g_7 is replaced by an infrequent subgraph which contains nodes "S", "O" and an edge between them, nearly 81% edges will be covered. Motivated by this, in this paper we study the top-k edge-diversified patterns discovery problem, which finds k subgraphs from a graph database such that a maximum number of edges in the graph database can be covered. The following example illustrates top-k edge-diversified patterns and their benefits.

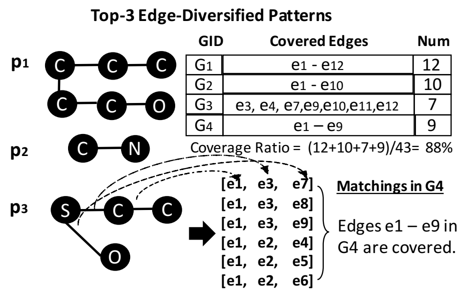
EXAMPLE 2. *Reconsider Example 1. Figure 3 presents top-3 edge-diversified patterns (*i.e., *p_1, p_2 and p_3) for the sample database in Figure 1. Only p_1 is a frequent subgraph while p_3 (resp. p_2) exists in G_4 (resp. G_3) only. The matchings of p_3 in G_4 are also shown in this figure where all 9 edges in G_4 are covered. As such, top-3 edge-diversified patterns can cover almost 88% edges in all 4 graphs in the sample database.* ■


Some Frequent Subgraphs (sup_min = 0.3)
Top-3 Diversified Frequent Subgraphs
g1
g2
g3
g4
g5
g6
g7
g8
......
(a)
(b)

GID	Covered Edges	Num
G1	e1 - e12	12
G2	e1 - e10	10
G3	e1, e2, e6	6
G4	e1	1

Coverage Ratio = (12+10+6+1)/43= 67%

Figure 2: Frequent subgraphs and diversified frequent subgraphs.


Top-3 Edge-Diversified Patterns
p1
p2
p3
Matchings in G4
Edges e1 – e9 in G4 are covered.

GID	Covered Edges	Num
G1	e1 - e12	12
G2	e1 - e10	10
G3	e3, e4, e7,e9,e10,e11,e12	7
G4	e1 – e9	9

Coverage Ratio = (12+10+7+9)/43= 88%

[e1, e3, e7]
[e1, e3, e8]
[e1, e3, e9]
[e1, e2, e4]
[e1, e2, e5]
[e1, e2, e6]

Figure 3: Top-k edge-diversified patterns.

The following three examples show the wide application potential of top-k edge-diversified patterns.

- **Visual Subgraph Query Formulation.** Declarative query languages formulating subgraph queries in textual form are not friendly to non-programmers or database experts. Top-k edge-diversified patterns can support interactive query construction through a visual query interface (*also known as* GUI). A case study in Section 7 exhibits its advantages in query formulation over existing approaches [17, 18].
- **Exploratory Subgraph Search.** Users without clear intent or familiarity of a graph database often prefer exploratory search over static queries [19, 20]. As can be seen in the case study (Section 7), top-k edge-diversified patterns can facilitate exploratory subgraph search, as it summarizes resulted graphs and helps users with better understanding their search intent.
- **Drug Repositioning.** Discovering therapeutic drugs from a drug set such as DRUGBANK [4] (*i.e.,* drug repositioning) can target altered signaling pathways within a cell to restore the physiological state of a disease network [21]. Top-k edge-diversified patterns can facilitate drug repositioning by checking if a drug can target on the patterns, as a candidate drug is supposed to target more edges (*i.e.,* signaling pathways) in the disease network.

However, *top-k edge-diversified patterns discovery* problem introduces non-trivial challenges. First, it is NP-hard. Second, adapting existing subgraph enumeration and frequent subgraph mining techniques to this problem is inadequate since they fall short in handling large databases or guaranteeing patterns' quality. In this paper, we address these challenges and propose a novel solution for top-k edge-diversified patterns discovery problem. We make the following contributions.

[4]https://go.drugbank.com/

Table 1: List of key notations.

Notation	Description
g, G, D	a subgraph, a graph, a graph database
$V(G), E(G)$	vertex set of graph G, edge set of graph G
$p, \mathcal{P}$	a pattern, a pattern set
k	number of edge-diversified patterns
sup_{min}	minimum support
$Cov(G_i, G_j)$	cover set of G_i over G_j
$\lvert Cov(G_i, G_j)\rvert$	coverage of G_i over G_j
E_{max}	maximum number of edges in an edge-diversified pattern
S, S_{fre}	a set of subgraphs, a set of frequent subgraphs

- To the best of our knowledge, we are the first to study *top-k edge-diversified patterns discovery problem* and address it with two adapted baseline solutions.
- We further present a generic and extensible framework called TED for this problem which requires limited memory and achieves a guaranteed approximation ratio.
- Three optimization strategies are developed to improve the performance and a lightweight version called TEDLITE is designed to handle even larger graph databases.
- By using real-world data graph repositories, extensive experimental evaluations are provided to show the superiority of our methods over two baseline solutions.

The rest of this paper is organized as follows. In Section 2, we provide some preliminaries and the problem statement. In Section 3, two baseline solutions are proposed for top-k edge-diversified patterns discovery problem. In Section 4, our proposed novel framework TED is presented, followed by three optimization strategies in Section 5 and a lightweight version TEDLITE in Section 6. Section 7 shows extensive experimental results. Related work are in Section 8 and conclusions are made in Section 9. Formal proofs of all theorems and lemmas are given in [1].

2 PRELIMINARIES

Table 1 lists the notations and acronyms used in this paper.

2.1 Key Concepts

A simple graph G is represented as $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. In this paper, we assume undirected simple graph whose vertex $v \in V$ is labeled with $l(v)$ and edge $e \in E$ is labeled with $l(e)$. [5] In this paper, we focus on a graph database containing a large collection of small- or medium-sized graphs (denoted as D). A unique *index* (*i.e.,* ID) is assigned to each graph in D. We denote a graph with index i as $G_i \in D$.

DEFINITION 1 (**SUBGRAPH ISOMORPHISM**). Given two graphs G_1 and G_2, a subgraph isomorphism is an injection $f \colon V(G_1) \to V(G_2)$ such that 1) $\forall v \in V(G_1)$, $l(v) = l'(f(v))$ and 2) $\forall (u, v) \in E(G_1)$, $(f(u), f(v)) \in E(G_2)$ and $l(u, v) = l'(f(u), f(v))$ where l and l' are the labeling functions of graph G_1 and G_2, respectively.

G_1 is *subgraph isomorphic to* G_2 if there exists at least one subgraph isomorphism f from G_1 to G_2. We also say that G_2 is *covered* by G_1 or that G_2 *contains* G_1 (denoted by $G_1 \subseteq G_2$). Given a subgraph isomorphism f and the subgraph G' of G_2 consisting of vertices $f(v)$ and edges $(f(u), f(v))$ where $u, v \in V(G_1)$, we say that G' is a **matching** of G_1 in G_2. We may simply say that G' is a matching. The edge $(f(u), f(v))$ is called covered edge.

[5]For the graph with labeled vertex and unlabeled edge, each edge $e \in E$ is labeled with the concatenation of labels (*i.e.,* a new label joining two labels together) of its two end vertices (*i.e.,* $l(e) = l(u).l(v)$).

Definition 2 (**Cover Set and Coverage**). Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, if G_1 is subgraph isomorphic to G_2 and the matchings are $\mathcal{F}$, the cover set of G_1 over G_2 is $Cov(G_1, G_2) = \cup_{f \in \mathcal{F}} (f(u), f(v))$ and the coverage is $|Cov(G_1, G_2)|$.

Observe that the cover set of G_1 over G_2 is union of all covered edges. The coverage is cardinality of the cover set. In addition, the cover set of a set of graphs $\mathcal{G}_1 = \{G_1, G_2, ..., G_i, ..., G_n\}$ over $\mathcal{G}_2 = \{G'_1, G'_2, ..., G'_j, ..., G'_m\}$ is defined as $Cov(\mathcal{G}_1, \mathcal{G}_2) = \cup_i \cup_j Cov(G_i, G'_j) = \cup_j \cup_i Cov(G_i, G'_j)$ where $i \in [1, n]$ and $j \in [1, m]$.

Example 3. *The graph g_7 in Figure 2 is subgraph isomorphic to graph G_1 in Figure 1. There are four matchings, $[e_7, e_9]$, $[e_7, e_{10}]$, $[e_8, e_{11}]$, and $[e_8, e_{12}]$. Thus, the covered edges are $e_7, e_8, e_9, e_{10}, e_{11}$, and e_{12}. The cover set is therefore $\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$ and the coverage is $|\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}| = 6$.* ■

2.2 Problem Statement

Definition 3 (**Top-k Edge-Diversified Patterns Discovery**). Given a graph database $D = \{G_1, G_2, \cdots, G_j, \cdots, G_n\}$ and an integer k, top-k edge-diversified patterns discovery is to find k patterns $\mathcal{P} = \{p_1, p_2, \ldots, p_i, \ldots p_k\}$ from D such that the total coverage of $\mathcal{P}$ over D (denoted by $|Cov(\mathcal{P}, D)|$), *i.e.,* $|\cup_i \cup_j Cov(p_i, G_j)|$, is maximized, where $Cov(p_i, G_j)$ is the cover set of p_i over G_j, and the number of edges of p_i (*i.e.,* $|E(p_i)|$) is no more than a given size threshold E_{max}.

Remark. Note that $|E(p_i)|$ is not allowed to be larger than a given threshold E_{max}, and the reasons are twofold. First, if some graphs in D are apparently larger than other graphs, there is a likelihood that selecting those graphs as the top-k edge-diversified patterns is already an optimal solution. Thus, we don't have to discuss the problem here. Second, in most applications, such as visual subgraph query formulation, patterns should not be too large [17], since pattern budget (e.g., minimum size, maximum size and number of patterns) is pre-defined for pattern mining. Moreover, the problem can be reformed and adapted to various applications by imposing additional constrains such as minimum size on the patterns, which will be further elaborated in Section 7 (Exp 7).

Theorem 1. *The top-k edge-diversified patterns discovery problem is NP-hard.*

3 BASELINE SOLUTIONS

In this section, we present two baseline solutions for this problem. The first one, named as ALL_g, is to enumerate all subgraphs S from the database D, and then conduct a greedy search. Given a minimum support threshold sup_{min}, the second solution named as FSG_g first generates all frequent subgraphs S_{fre} whose support are no less than sup_{min}, instead of all subgraphs. It then adopts the same greedy strategy as ALL_g to find top-k edge-diversified patterns.

3.1 Baseline Solution ALL_g

By adopting greedy search, we come up with the first baseline solution ALL_g, whose pseudo-code is shown in Algorithm 1. Given a graph database $D = \{G_1, G_2, ...G_n\}$ and an integer k, it first enumerates all subgraphs $S = \{s_1, s_2, ...\}$ from D such that $|E(s_i)| \leq E_{max}$

Algorithm 1 Baseline solution ALL_g

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-$k$ edge-diversified patterns
1: $S \leftarrow$ EnumAllSub$(D, E_{max})$
2: $\mathcal{P} \leftarrow$ MaxCover$(S, k)$                ▷ call procedure MaxCover
3: return $\mathcal{P}$
4: procedure MaxCover$(S, k)$
5:     $\mathcal{P} \leftarrow \phi$
6:     for iter = 1 to k do
7:         $p \leftarrow argmax_{p' \in S} |Cov(p', D) \setminus Cov(\mathcal{P}, D)|$
8:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}, S \leftarrow S \setminus p$
9:     return $\mathcal{P}$
```

Algorithm 2 Baseline solution FSG_g

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Heuristic top-k edge-diversified patterns
1: $S_{fre} \leftarrow$ EnumFreSub$(D, E_{max})$
2: $\mathcal{P} \leftarrow$ MaxCover$(S_{fre}, k)$                ▷ MaxCover in Algorithm 1
3: return $\mathcal{P}$
```

(Line 1), by using an existing subgraph enumeration method [2–6]. Then, the procedure MaxCover is invoked to generate top-k edge-diversified patterns (Line 2). MaxCover follows a iterative procedure (Lines 6-8). In each iteration, it greedily selects a pattern p from S such that the cover set of p contains a maximum number of uncovered edges, *i.e.,* $|Cov(p', D) \setminus Cov(\mathcal{P}, D)|$ (Line 7). Once the resulting pattern p is selected, it will be removed from S to pattern set $\mathcal{P}$ (Line 8).

Observe that MaxCover follows the same greedy strategy used for solving the max k-cover problem [22]. Therefore, it can achieve an approximation ratio of $1 - \exp(-1)$.

Lemma 1. *Worst case time and space complexities of ALL_g are $O(|D|2^{max(V(G))^2} + k|S||D|max(V(G))^{E_{max}})$ and $O(max(E(G))|D| + E_{max}|S|)$, respectively, where $max(V(G))$ (resp. $max(E(G))$) is maximum number of vertices (resp. edges) in graph $G \in D$, and $|D|$ (resp. $|S|$) is number of graphs in D (resp. S).*

3.2 Baseline Solution FSG_g

Note that ALL_g requires searching all subgraphs which may incur large computational overload. To address this, we further propose the second baseline solution FSG_g, whose procedure is shown in Algorithm 2. Instead of enumerating all subgraphs from graph database D, FSG_g first adopts existing frequent subgraph mining methods [7–12] to generate frequent subgraphs S_{fre} (Line 1). Then MaxCover in Algorithm 1 is invoked to find a pattern set on S_{fre} (Line 2).

Since only frequent subgraphs instead of all subgraphs are selected from the underlying database for generating final patterns, the computational cost of FSG_g is thus reduced.

Lemma 2. *Worst case time and space complexities of FSG_g are $O(|D|2^{max(V(G))^2} + k|S_{fre}||D|max(V(G))^{E_{max}})$ and $O(max(E(G))|D| + E_{max}|S_{fre}|)$, respectively, where $max(V(G))$ (resp. $max(E(G))$) is maximum number of vertices (resp. edges) in graph $G \in D$, and $|D|$ (resp. $|S_{fre}|$) is number of graphs in D (resp. S_{fre}).*

3.3 Limitations of Baseline Solutions

Observe that ALL_g provides an approximation ratio of $1 - \exp(-1)$ for final patterns, but it is computationally challenging. Compared to ALL_g, FSG_g significantly reduces the computational cost by selecting frequent subgraphs, based on which the final patterns are derived. However, the pattern quality is not guaranteed since no approximation bound can be provided as only frequent subgraphs are considered. Moreover, they suffer from the following problems:

(1) *Excessive memory consumption.* Both ALL$_g$ and FSG$_g$ need to store all subgraphs or a subset of all subgraphs in memory. However, the number of subgraphs in a graph database is exponential in database size. As a graph database continues to grow rapidly in size, storing those subgraphs in memory will lead to excessive memory consumption. As will be shown in Section 7 (Fig. 10, Exp 2), even the subgraphs of 5,000 graphs in PubChem database are not easy to be kept in memory.
(2) *A mass of unnecessary computations.* Since the (frequent) subgraph enumeration and MaxCover procedure are performed sequentially, all (frequent) subgraphs, no matter whether they can improve the total coverage of final patterns $\mathcal{P}$, are computed and stored before invoking MaxCover procedure. Therefore, there are a mass of unnecessary computations.

4 TED: A NOVEL FRAMEWORK

To address aforementioned limitations, we propose a novel framework called Ted (*i.e.,* Top-*k* Edge-Diversified patterns discovery). The main idea is to integrate the search process for top-k patterns into subgraph enumeration. Specifically, it maintains only k candidate patterns $\mathcal{P}$ in memory. These candidates are supposed to potentially maximize the total coverage. When a new subgraph g is enumerated and considered as a *promising candidate* (see Sec. 4.1), it will be added to $\mathcal{P}$ by swapping out one of the patterns. Compare to the above two baseline solutions, Ted has the following advantages:

(i) *Limited memory consumption.* Ted always maintains only k patterns in memory to avoid excessive memory consumption for storing all subgraphs. As can be seen in Theorem 2, the space complexity of Ted is $O(max(E(G))|D|)$ which is far less than that of ALL$_g$ ($O(max(E(G))|D| + E_{max}|S|)$, Lemma 1) and FSG$_g$ ($O(max(E(G))\ |D| + E_{max}|S_{fre}|)$, Lemma 2).
(ii) *Guaranteed pattern quality.* Ted can achieve an approximation bound of 1/4 and better performance in experimental study. To this end, it provides a swapping criteria such that each newly generated subgraph g is deliberately swapped with one existing pattern in $\mathcal{P}$.
(iii) *Effective pruning strategies.* Ted is able to prune subgraph search space by integrating the search process for top-k patterns into subgraph enumeration.
(iv) *Desirable scalability.* Ted and its lightweight version, namely TedLite, are capable of handling large databases.

In what follows, we begin with a basic Ted algorithm (denoted by Ted_base) in this section, which can achieve both (i) and (ii). Then, three optimization strategies are developed to further realize (iii) (Section 5). Finally, we present TedLite, which is a lightweight version of Ted and able to accomplish (iv) (Section 6).

4.1 The Basic Ted Algorithm

The basic Ted algorithm (Ted_base) generates top-k edge-diversified patterns by alternately performing subgraph enumeration and search process for top-k patterns. For subgraph enumeration, it adopts a depth-first search (DFS) strategy to traverse the search space. For example, $g_{1,1}, g_{2,1}, g_{3,1}$, and $g_{4,1}$ in Figure 4 are traversed

Algorithm 3 A basic Ted algorithm (Ted_base)

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-k edge-diversified patterns
1: $\mathcal{P} \leftarrow \phi$
2: $S_p \leftarrow$ EnumSub$(D, |E| = 1)$
3: for each $g \in S_p$ do
4:     $S_p \leftarrow S_p \setminus g$
5:     $\mathcal{P} \leftarrow$ PatternMaintain$(\mathcal{P}, g, D)$
6:     $\mathcal{P}_g \leftarrow$ RightMostExtend$(g, D, E_{max})$
7:     $S_p \leftarrow \mathcal{P} \cup \mathcal{P}_g$
8: procedure PatternMaintain$(\mathcal{P}, g, D)$
9:     if $|\mathcal{P}| < k$ then
10:        $\mathcal{P} \leftarrow \mathcal{P} \cup \{g\}$
11:        return $\mathcal{P}$
12:    Score$_L$, $p_t \leftarrow$ min(genLossScore$(\mathcal{P})$)
13:    Score$_B \leftarrow$ genBenefitScore$(g)$
14:    if Score$_B > (1+\alpha)$Score$_L + \frac{(1-\alpha)|Cov(\mathcal{P},D)|}{k}$ then
15:        $\mathcal{P} \leftarrow \mathcal{P} \setminus p_t \cup \{g\}$
16:    return $\mathcal{P}$
```

sequentially. Although Apriori-like approaches [7, 8] that adopt breadth-first search (BFS) strategy have been widely studied, they require generating a lot of duplicated candidates and testing subgraph isomorphism. In contrast, depth-first search (DFS) strategy combines the pattern generating and isomorphism checking into one process to address this problem. For top-k patterns search on the enumerated subgraphs, a swapping-based strategy is adopted for maintaining patterns with limited memory consumption.

Ted_base is outlined in Algorithm 3. Given a graph database $D = \{G_1, G_2, ...G_n\}$ and an integer k, it first enumerates all 1-sized subgraphs (*i.e.,* edges) and appends them into the set S_p (Line 2). Then, an iterative process (Lines 3-7) is performed to generate final patterns $\mathcal{P}$ by taking S_p and D as inputs. Specifically, each subgraph $g \in S_p$ is first considered and removed from S_p (Lines 3-4). Then, the procedure PatternMaintain is performed to update top-k edge-diversified patterns $\mathcal{P}$ with newly enumerated subgraph g (Line 5). After that, the right-most extension method (Procedure RightMostExtend) extends each subgraph $g \in S_p$ with one more edge (Line 6) so that its supergraphs will be considered in the next iteration. The process repeats until S_p is empty. More details about the two main procedures, PatternMaintain and RightMostExtend, are discussed below.

4.1.1 Pattern Maintenance (PatternMaintain).

As discussed above (Section 3), the max k-cover problem is a subproblem of top-k edge-diversified patterns discovery. However, greedy-search based solutions typically find entire subgraphs and store them in memory and hence cannot be effectively exploited for large database. To address this, PatternMaintain maintains only k patterns in memory with a swapping-based method motivated by existing maximum coverage solver in the context of streaming scenario [23–25]. We first introduce the concepts of *loss score* and *benefit score* below to facilitate exposition.

Definition 4 (**Loss score**). Given a pattern set $\mathcal{P}$ and a database D, the **loss score** of a pattern $p \in \mathcal{P}$ is the decrease of total coverage caused by removing p from $\mathcal{P}$, i.e.,

$$\text{Score}_L(p, \mathcal{P}, D) = |\cup_{p \in \mathcal{P}} Cov(p, D) \setminus \cup_{p' \in \mathcal{P} \setminus p} Cov(p', D)|.$$

Definition 5 (**Benefit score**). Given a pattern set $\mathcal{P}$ and a database D, the **benefit score** of a pattern $g \notin \mathcal{P}$ is the increase of total coverage caused by adding g to $\mathcal{P}$, i.e.,

$$\text{Score}_B(g, \mathcal{P}, D) = |\cup_{p' \in \mathcal{P} \cup \{g\}} Cov(p', D) \setminus \cup_{p \in \mathcal{P}} Cov(p, D)|.$$

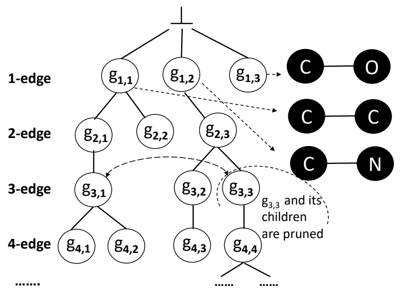


Figure 4: The DFS search space.

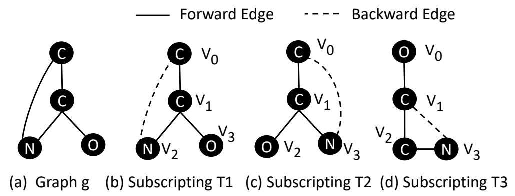


Figure 5: A graph G (a) and its subscriptings (b-d).

PatternMaintain first greedily selects k patterns into the pattern set $\mathcal{P}$ (Lines 9-11, Algorithm 3). When a new subgraph g is generated, a swapping-based process is developed to determine if g should be swapped into $\mathcal{P}$ (Lines 12-15). Specifically, it first calculates and ranks loss scores for each pattern $p \in \mathcal{P}$, and then records the pattern p_t and its pattern score Score_L such that p_t has a minimum loss score (Line 12). Meanwhile, the benefit score Score_B of g is also recorded (Line 13). The subgraph g is considered as a **promising candidate** and swapped into $\mathcal{P}$ (Lines 14-15) if the following *swapping criteria* is satisfied:

$$\text{Score}_B > (1+\alpha)\text{Score}_L + (1-\alpha)|Cov(\mathcal{P}, D)|/k \qquad (1)$$

where $\alpha \in [0, 1]$ is a *swapping threshold* for balancing the loss score Score_L and average coverage of patterns in $\mathcal{P}$ (*i.e.,* $|Cov(\mathcal{P}, D)|/k$). Note that there are three variants of the swapping criteria, namely, $Swap_1$ [23], $Swap_2$ [24], and $Swap_\alpha$ [25] where $\alpha = 1$, $\alpha = 0$, and $\alpha \in (0, 1)$, respectively. By default, we set $\alpha = 1$ and compare it with other variants in Section 7. The pattern p_t is swapped out once g is swapped in. The pattern set $\mathcal{P}$ is hence updated (Line 15).

4.1.2 Right-Most Extension (RightMostExtend). TED_base adopts a depth-first search (DFS) strategy to enumerate all possible subgraphs by iteratively performing right-most extension (*i.e.,* RightMostExtend). Figure 4 outlines the DFS search space, where each node represents an m-edge graph (denoted by $g_{m,i}$, *i.e.,* i-th graph with m edges). Each link between two nodes represents a possible right-most extension, *i.e.,* a desirable way to extend m-edge graph to $(m+1)$-graph, which is first proposed in gSpan [10]. For example, $g_{1,1}, g_{2,1}, g_{3,1}$, and $g_{4,1}$ in Figure 4 are traversed sequentially. $g_{4,1}$ is a right-most extension of $g_{3,1}$. We begin with the DFS subscripting technique for encoding a graph to illustrate right-most extension.

Dfs Subscripting. Given a graph g, we can perform DFS on it and generate the corresponding DFS trees. For example, for graph g in Figure 5(a), there are at least three DFS trees as the solid lines show in Figure 5(b)-(d). The subscript of each vertex in a DFS tree is sequentially built in the process of DFS traversal. That is, V_i is visited before V_j if $i < j$. As a result, each DFS tree T can be represented as vertices sequence. The DFS tree for Figure 5(b) is (V_0, V_1, V_2, V_3). The graph g subscripted with a DFS tree T is denoted by g_T and hence T is named a DFS subscripting of g. Given g_T, *root vertex* is the first visited vertex (*i.e.,* V_0) and *right-most vertex* is the last visited vertex (*i.e.,* $V_{|V(g)|-1}$). The *right-most path* is then defined as the straight path from root vertex to right-most vertex. For example, Figure 5(b)-(d) present three DFS subscriptings whose root vertices are all V_0 and right-most vertices are V_3. The right-most path for Figure 5(b) (resp. (c)) and (d) are (V_0, V_1, V_3) and (V_0, V_1, V_2, V_3), respectively.

Moreover, the *forward edge* contains all edges in the DFS tree T (denoted by E_T^f). The *backward edge* consists of all other edges (denoted by E_T^b). It is obvious that $(V_i, V_j) \in E_T^f$ if $i < j$, and $(V_i, V_j) \in E_T^b$ otherwise. In Figure 5(b), (V_0, V_1) and (V_2, V_0) are a forward edge and backward edge, respectively.

Right-Most Extension. The straightforward method for extending an m-edge graph to an $(m+1)$-edge graph is to add a new edge to any position of the m-edge graph. It is inefficient since there are m positions for extending which will incur a lot of duplicated graphs. Right-most extension is an efficient method to extend a graph with one more edge by restricting the extension positions.

Definition 6 (**Right-most extension**). Given an m-edge graph g and a DFS tree T, right-most extension is to extend g to an $(m+1)$-edge graph g' with an edge e such that at least one of the following rules is satisfied: 1) *forward extension.* e is extended from vertices on the right-most path such that an additional vertex is introduced and 2) *backward extension.* e is extended from right-most vertex to other vertices on the right-most path.

For example, given the graph g in Figure 5(a) and the DFS tree T in Figure 5(b), the forward extension of an edge e can be edges extending from either V_0, V_1 or V_3 to a newly introduced vertex. The backward extension should be (V_3, V_0). Since each graph has various subscriptings which will cause multiple extensions, the subscripting with minimum lexicographic order is selected as base subscripting for extending [10]. As a result, the node (*e.g.,* $g_{3,3}$ in Figure 4) and its descendants in the search space are pruned if the subscripted graph in the node has non-minimum subscripting.

Remark. The completeness of results generated by right-most extension is guaranteed [10]. As will be seen later (Theorem 4), the quality of final patterns (*i.e.,* coverage) is theoretically guaranteed.

4.2 Fast Pattern Maintenance

A naive method for computing loss score (Line 12, Algorithm 3) and benefit score (Line 13, Algorithm 3) is to directly calculate cover set $Cov(\cdot)$ according to their definitions, which is obviously inefficient. In this section, we propose the FastMaintain algorithm (Algorithm 4) to facilitate fast pattern maintenance. To begin with, the Private-Edge-Set Index (denoted by PES-Index) is developed to accelerate loss score and benefit score computation. PES-Index consists of five components:

- $|Cov(\mathcal{P})|$: total coverage of $\mathcal{P}$ over D (*i.e.,* $|Cov(\mathcal{P}, D)|$).
- $|pCov(p)|$: *private coverage* of p. $pCov(p)$ is the set of edges in D, which is in the cover set of p but not in the cover set of $p' \in \mathcal{P} \setminus p$, *i.e.,* $pCov(p) = Cov(p, D) \setminus Cov(\mathcal{P} \setminus p, D)$.

- $rCov(e)$: *reverse cover set* of an edge e. It refers to a subset of patterns that contain e in the cover set. That is, $rCov(e) = \{p|p \in \mathcal{P}, e \in Cov(p, D)\}$.
- $rCnt(i)$: *reverse counting set* of number of edges. It refers to a set of patterns such that each pattern p's private coverage is i. That is, $rCnt(i) = \{p|p \in \mathcal{P}, |pCov(p)| = i\}$ where $i \in [0, \sum_{G \in D} |E(G)|]$.
- p_{min}: the pattern $p \in \mathcal{P}$ with minimum private coverage $|pCov(p)|$.

Moreover, four operations, INSERT, DELETE, UPDATE and SELECT, are developed on PES-Index. Specifically, INSERT operation is to insert a newly enumerated subgraph into current pattern set and update PES-Index; DELETE aims to remove a pattern from current pattern set and update PES-Index; UPDATE is a combination operation that sequentially calls DELETE and INSERT operations; SELECT supports fast computations for loss score and benefit score.

Given a graph database D, a pattern set $\mathcal{P}$ and each enumerated graph g, if the number of existing patterns is less than k, INSERT operation is performed to update $\mathcal{P}$ (Line 1, Algorithm 4), and g is directly inserted into $\mathcal{P}$ (Line 8). For each edge e covered by g (*i.e.,* $e \in Cov(g, D)$), its reverse cover set is updated by adding g (Lines 9-10). If its reverse cover set only contains g (*i.e.,* $|rCov(e)|$==1), it indicts that e is only covered by g. Hence, both g's private coverage $|pCov(g)|$ and the total coverage $Cov(\mathcal{P})$ increase by 1 (Line 12). If its reverse cover set contains another pattern p, the private coverage of p (*i.e.,* $|pCov(p)|$) should be decreased by 1 since e is also covered by g (Line 14). Once $|pCov(p)|$ is updated, the reverse counting set $rCnt(i)$ is updated by moving p from $rCnt(|pCov(p)| + 1)$ to $rCnt(|pCov(p)|)$ (Line 15). The process repeats until all $e \in Cov(g, D)$ are considered. Then, $rCnt(|pCov(g)|)$ is also updated (Line 16). Based on all computed $rCnt(i)$, we can directly figure out which pattern has minimum loss score by selecting one pattern p_{min} in $rCnt(i) \neq \phi$ such that i is minimum (Line 17). With the constructed PES-Index, the minimum loss score (*i.e.,* SCORE$_L$) and the corresponding pattern (*i.e.,* p_t) can be easily obtained (SCORE$_L$ = $|pCov(p_{min})|$ and $p_t = p_{min}$, Lines 2 and 27). The benefit score SCORE$_B$ is calculated by counting the cases where the reverse cover set $|rCov(e)|$ of $e \in Cov(g, D)$ is 0 (Lines 3 and 28), as $|rCov(e)| = 0$ indicates that e is not covered by existing patterns.

If the swapping criteria is satisfied (Line 4), the UPDATE operation is performed by sequentially calling DELETE operation for p_t and INSERT operation for g (Line 5). Firstly, p_t is directly removed from $\mathcal{P}$ and $rCnt(|pCov(p_t)|)$ (Line 19). For each edge $e \in Cov(p_t, D)$, the corresponding $rCov(e)$ should be maintained by removing p_t, and the total coverage should be decreased by 1 if $|rCov(e)|$ is 0 (Lines 20-22). In addition, if the $|rCov(e)|$ of e contains only one pattern p, the $|pCov(p)|$ should be increased by 1, and the reverse counting set $rCnt(i)$ should be updated by moving p from $rCnt(|pCov(p)| - 1)$ to $rCnt(|pCov(p)|)$ (Lines 23-25).

Remark. Note that PES-Index is motivated by the existing PNP-Index [44], which was designed for diversified top-k clique search. The main difference lies in that PNP-Index is to build the relationship between the enumerated clique and the vertices in a single large data graph. Each clique is exactly one matching in the data graph and only the vertices in the matching are indexed. In contrast, PES-Index aims to index the cover set (*i.e.,* a set of edges) of a particular edge-diversified pattern over a graph database containing a set of graphs. Each pattern here has multiple matchings in a graph.

Algorithm 4 Fast pattern maintenance (FASTMAINTAIN)

```
Input: Graph database $D = \{G_1, G_2, ...G_n\}$, existing patterns $\mathcal{P}$, graph $g$
Output: Updated patterns
 1: if $|\mathcal{P}| < k$, $\mathcal{P} \leftarrow$ INSERT($g$), return $\mathcal{P}$
 2: SCORE$_L$, $p_t \leftarrow$ SELECT($g, true$)
 3: SCORE$_B \leftarrow$ SELECT($g, false$)
 4: if SCORE$_B > (1 + \alpha)$SCORE$_L + \frac{(1-\alpha)|Cov(\mathcal{P})|}{k}$ then
 5:     $\mathcal{P} \leftarrow$ UPDATE($p_t, g$)
 6: return $\mathcal{P}$
 7: procedure INSERT($g$)
 8:     $\mathcal{P} \leftarrow \mathcal{P} \cup \{g\}$
 9:     for $e \in Cov(g, D)$ do
10:         $rCov(e) \leftarrow rCov(e) \cup \{g\}$
11:         if $|rCov(e)|$==1 then
12:             $|pCov(g)| \leftarrow |pCov(g)| + 1$; $|Cov(\mathcal{P})| \leftarrow |Cov(\mathcal{P})| + 1$
13:         if $|rCov(e)|$==2 then
14:             for $p \in rCov(e) \setminus g$, $|pCov(p)| \leftarrow |pCov(p)| - 1$
15:             move $p$ from $rCnt(|pCov(p)| + 1)$ to $rCnt(|pCov(p)|)$
16:     $rCnt(|pCov(g)|) \leftarrow rCnt(|pCov(g)|) \cup \{g\}$
17:     $p_{min} \leftarrow$ a pattern in $rCnt(i) \neq \phi$ s.t. $i$ is minimum.
18: procedure DELETE($p_t$)
19:     remove $p_t$ from $\mathcal{P}$ and $rCnt(|pCov(p_t)|)$
20:     for $e \in Cov(p_t, D)$ do
21:         $rCov(e) \leftarrow rCov(e) \setminus p_t$
22:         $|Cov(\mathcal{P})| \leftarrow |Cov(\mathcal{P})| - 1$, if $|rCov(e)|$==0
23:         if $|rCov(e)|$==1 then
24:             for $p \in rCov(e)$, $|pCov(p)| \leftarrow |pCov(p)| + 1$
25:             move $p$ from $rCnt(|pCov(p)| - 1)$ to $rCnt(|pCov(p)|)$
26: procedure SELECT($g, flag$)
27:     if $flag$ is true, return $|pCov(p_{min})|, p_{min}$
28:     else return $|\{e|e \in Cov(g, D)$ and $|rCov(e)| == 0\}|$
```

EXAMPLE 4. *Let $\alpha = 1$, $k = 3$, and the current pattern set $\mathcal{P} = \{g_1, p_1, p_3\}$ where g_1 and p_1 (resp. p_3) are shown in Figure 2(a) and Figure 3, respectively. $|pCov(g_1)|$, $|pCov(p_1)|$ and $|pCov(p_3)|$ are 2, 10, and 8, respectively, where $pCov(g_1) = \{G_3 : e_2, e_6\}$ (*i.e., *e_2 and e_6 of G_3). In addition, $|Cov(\mathcal{P})| = 33$ and $rCnt(2) = \{g_1\} \neq \phi$. Hence, $p_t = g_1$ and the minimum loss score SCORE$_L$ = $|pCov(g_1)| = 2$. When the graph p_2 (Figure 3) is newly enumerated, its benefit score SCORE$_B$ = 7, as $Cov(p_2, D) = \{G3 : e_3, e_4, e_7, e_9 - e_{12}\}$ and $|\{e|e \in Cov(p_2, D)$ and $|rCov(e)| == 0\}| = 7$. Since SCORE$_B > (1 + \alpha)$SCORE$_L = 4$, g_1 should be removed from $\mathcal{P}$ and then $|Cov(\mathcal{P})| = 31$. In addition, p_2 should be added into $\mathcal{P}$, *i.e.,* $\mathcal{P} = \{p_1, p_2, p_3\}$. The total coverage is $|Cov(\mathcal{P})| = 38$ as shown in Figure 3.* ∎

THEOREM 2. *Worst case time and space complexities of TED_BASE are $O(|D|2^{max(V(G))^2})$ and $O(max(E(G))|D|)$, respectively, where $max(V(G))$ (resp. $max(E(G))$) is maximum number of vertices (resp. edges) in graph $G \in D$, and $|D|$ is number of graphs in D.*

5 OPTIMIZATIONS

Recall that TED_BASE integrates the search process for top-k patterns into subgraph enumeration, which guarantees the pattern quality with limited memory consumption. However, it does not explore the potentials of reducing the search space and unnecessary computations to boost the overall effectiveness. Therefore, in this section, we further propose three optimization strategies, namely, *Dynamic Support Setting* in Section 5.1, *Promising Right-Most Extension* in Section 5.2, and *Initial Pattern Selection* in Section 5.3. In short, *Dynamic Support Setting* directly filters unpromising patterns with lower support. While *Promising Right-Most Extension* prunes these unpromising patterns based on the covering relationship of a graph and its descendants instead of support. *Initial Pattern Selection* targets on promoting the quality of initial patterns so that better pruning power are introduced in the early stage.

Algorithm 5 An improved TED algorithm (TED_DSS)

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-k edge-diversified patterns
1: $\mathcal{P} \leftarrow \phi$
2: $S_p \leftarrow$ EnumSub($D$, $|E| = 1$)
3: for each $g \in S_p$ do
4:     $S_p \leftarrow S_p \setminus g$
5:     if DSS($\mathcal{P}$) > COUNT($g$) then
6:         CONTINUE
7:     $\mathcal{P} \leftarrow$ PatternMaintain($\mathcal{P}, g, D$)
8:     $\mathcal{P}_g \leftarrow$ RightMostExtend($g, D, E_{max}$)
9:     $S_p \leftarrow \mathcal{P} \cup \mathcal{P}_g$
```

5.1 Dynamic Support Setting

Observe that edge-diversified patterns may not be frequent as illustrated in Section 1. Therefore, for the sake of completeness of final results, TED_BASE considered each subgraph g even it is contained by only one graph $G \in D$. Hence, the search space is huge. Fortunately, we have the following observation:

OBSERVATION I. *The search space will be significantly reduced as the minimum support (sup_{min}) increases.*

The intuition behind the observation is that the support has anti-monotone property, *i.e.,* the support of a graph never exceeds that of its subgraphs. Therefore, if a graph with low support is definitely not a promising pattern, all descendants of this graph are filtered out. Motivated by this observation, *Dynamic Support Setting* (DSS) is developed. Specifically, it dynamically sets the value for minimum support sup_{min} to filter some unpromising subgraphs as necessary without sacrificing the completeness of the results. Let COUNT(g) be the number of graphs in D containing g, DSS adopts an iterative process to set minimum support: (1) Set a counter $cnt = 1$. (2) If COUNT(g) $= cnt$, the benefit score of g is no more than the maximum number of un-covered edges in cnt graphs, *i.e.,* $\max |\bigcup_{i \in \mathbb{I}}(E(G_i) \setminus Cov(\mathcal{P}, G_i))|$ where $\mathbb{I} \subseteq [1, |D|]$ and $|\mathbb{I}| = cnt$. If $\max |\bigcup_{i \in \mathbb{I}}(E(G_i) \setminus Cov(\mathcal{P}, G_i))|$ is less than $(1 + \alpha)$SCORE$_L$ + $(1 - \alpha)|Cov(\mathcal{P}, D)|/k$ (*i.e., g* is unpromising), $cnt = cnt + 1$, go to step (2). Go to step (3), otherwise. (3) Set minimum support $sup_{min} = cnt/|D|$.

Note that $\max |\bigcup_{i \in \mathbb{I}}(E(G_i) \setminus Cov(\mathcal{P}, G_i))|$ can be easily computed by selecting the first $|\mathbb{I}|$ values from a priority queue, where $|E(G_i) \setminus Cov(\mathcal{P}, G_i)|$ is in descending order. As such, DSS can be used to prune unpromising subgraphs by checking support only.

Based on DSS, an improved TED algorithm called TED_DSS, as shown in Algorithm 5, is developed. The difference between TED_DSS and TED_BASE lies in lines 5 and 6 where DSS strategy is introduced. That is, if COUNT(g) is less than DSS($\mathcal{P}$), *i.e.,* the product of minimum support sup_{min} and $|D|$, g is deemed as unpromising and will not be taken into account by pattern maintenance process.

5.2 Promising Right-Most Extension

Recall that in TED_BASE and TED_DSS, graphs are enumerated with right-most extension. As shown in Figure 6(a), an m-edge graph g is extended to an $(m + 1)$-edge graph g'. Then, g' is extended to an $(m + 2)$-edge graph g''. It is obvious they discard the relationship (see Observation II) between a subgraph and its descendants (*i.e.,* supergraphs).

OBSERVATION II. *The benefit score of a subgraph g' over graph G_i is at most the number of uncovered edges in G_i, and the edges in a graph G_i covered by g but not covered by g' will not be covered by g'' (*i.e., *a descendant of g').*

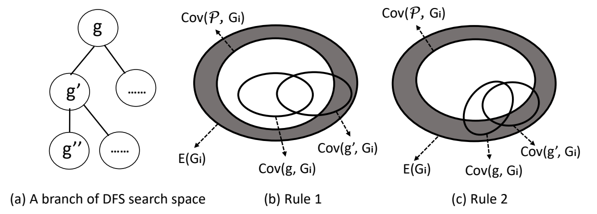


Figure 6: Illustration for PRM.

Algorithm 6 An improved TED algorithm (TED_PRM)

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-k edge-diversified patterns
1: $\mathcal{P} \leftarrow \phi$
2: $S_p \leftarrow$ EnumSub($D$, $|E| = 1$)
3: for each $g \in S_p$ do
4:     $S_p \leftarrow S_p \setminus g$
5:     $\mathcal{P} \leftarrow$ PatternMaintain($\mathcal{P}, g, D$)
6:     $\mathcal{P}_g \leftarrow$ RightMostExtend($g, D, E_{max}$)
7:     for $g' \in \mathcal{P}_g$ do
8:         if PRM($g, g'$) < $(1 + \alpha)$Score$_L$ + $\frac{(1-\alpha)|Cov(\mathcal{P},D)|}{k}$ then
9:             $\mathcal{P}_g \leftarrow \mathcal{P}_g \setminus g'$
10:    $S_p \leftarrow \mathcal{P} \cup \mathcal{P}_g$
```

Motivated by this observation, *Promising Right-Most Extension* (PRM) is developed by taking a prudent strategy to extend g to g' (and descendants of g', *e.g., g''*). Specifically, PRM extends g to g' if and only if one of the following PRM rules is satisfied.

DEFINITION 7 (**PRM RULES**). *1) Rule 1.* If $g \in \mathcal{P}$, g' extends from g with one edge if $|\cup_{i \in \mathbb{I}} (E(G_i) \setminus Cov(\mathcal{P}, G_i))| \geq (1 + \alpha)$SCORE$_L$ + $(1 - \alpha)|Cov(\mathcal{P}, D)|/k$, where $\mathbb{I}$ is the id set of graphs containing g. *2) Rule 2.* If $g \notin \mathcal{P}$, g' extends from g with one edge if $|\cup_{i \in \mathbb{I}} (E(G_i) \setminus (Cov(\mathcal{P}, G_i) \cup (Cov(g, G_i) \setminus Cov(g', G_i))))| \geq (1 + \alpha)$SCORE$_L$ + $(1 - \alpha)|Cov(\mathcal{P}, D)|/k$.

Intuitively, PRM rules is a formal explanation of Observation II. Its correctness is guaranteed by the following Theorem 3.

THEOREM 3. *Right-most extension with PRM rules has no effect on the quality (*i.e., *coverage) of final patterns.*

Owing to the pruning power of PRM rules, we propose an improved TED algorithm called TED_PRM, as shown in Algorithm 6, by introducing PRM into TED_BASE (Lines 7-9). Specifically, given an m-edge graph g and its all potential $(m + 1)$-edge supergraphs $\mathcal{P}_g$, for each $g' \in \mathcal{P}_g$, if g' does not satisfy PRM rules (Line 8), g' will be removed from $\mathcal{P}_g$ for further processing. Note that PRM process (Lines 7-9) can be integrated with RIGHTMOSTEXTEND (Line 6), both of them are individually listed here for ease of presentation.

5.3 Initial Pattern Selection

As discussed in Section 4.1, the initial k patterns are selected by traversing the search space in *DFS* manner (Lines 6, Algorithm 3). For example, suppose the search space is shown in Figure 4 and $k = 3$, the initial k patterns are $g_{1,1}$, $g_{2,1}$, and $g_{3,1}$. Observe that all of them contain the same substructure (*i.e., $g_{1,1}$*), they may cover the same edges in D. Therefore, the minimum loss score SCORE$_L$ may get small, so that the swapping criteria is easily satisfied (Line

Algorithm 7 An improved TED algorithm (TED_IPS)

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-k edge-diversified patterns
1: $\mathcal{P} \leftarrow$ IPS$(D, k)$
2: $S_p \leftarrow$ EnumSub$(D, |E| = 1)$
3: for each $g \in S_p$ do
4:     $S_p \leftarrow S_p \setminus g$
5:     $\mathcal{P} \leftarrow$ PatternMaintain$(\mathcal{P}, g, D)$
6:     $\mathcal{P}_g \leftarrow$ RightMostExtend$(g, D, E_{max})$
7:     $S_p \leftarrow \mathcal{P} \cup \mathcal{P}_g$
```

14, Algorithm 3). Although the swapping criteria is satisfied and a pattern g is hence swapped in, the pattern quality may be very low so that g will be swapped out finally. Obviously, frequently swapping patterns does harm to algorithm performance. To address this problem, we develop *Initial Pattern Selection* (IPS) technique motivated by the following observation.

OBSERVATION III. *The initial k patterns generated by traversing the search space in DFS manner share common substructure and lead to low loss score. While patterns generated with Breadth-First Search (*i.e., *BFS) on the search space tend to be structurally dissimlar to each other and may imply higher loss score.*

Specifically, IPS starts at the first node in first level (*e.g.,* $g_{1,1}$, Figure 4) of the search space. Then, it explores descendant nodes in second level (*e.g.,* $g_{2,1}$) if the extended graph (*e.g.,* $g_{2,1}$) has higher benefit score. The process repeats until no higher benefit score is obtained or the desirable number of edges (*i.e.,* E_{max}) is derived. As a result, the first pattern is obtained. After that, IPS adopts the same technique to generate patterns rooted at the other nodes in first level (*e.g.,* $g_{1,2}$ and $g_{1,3}$, Figure 4). Once all patterns rooted at nodes in first level are generated, the top-k patterns with maximum coverage are selected as initial pattern set.

The improved TED algorithm with IPS (denoted by TED_IPS) is developed as Algorithm 7. The only difference between TED_IPS and TED_BASE lies in line 1 where IPS strategy is introduced.

5.4 Putting Things Together

Algorithm 8 outlines the complete TED algorithm where all three optimizations are integrated. Specifically, IPS (Line 1) is introduced to improve the initial loss score. Then, DSS (Lines 5 and 6) utilizes minimum support to quickly prune unpromising patterns. Finally, PRM takes a prudent strategy to enumerate potential promising subgraphs (Lines 9-11). Theorem 4 ensures that the approximation ratio of our TED algorithm is lower bounded by 1/4.

Algorithm 8 TED algorithm (TED)

```
Input: graph database $D = \{G_1, G_2, ...G_n\}$, integer $k$, and $E_{max}$
Output: Near-optimal top-k edge-diversified patterns
1: $\mathcal{P} \leftarrow$ IPS$(D, k)$
2: $S_p \leftarrow$ EnumSub$(D, |E| = 1)$
3: for each $g \in S_p$ do
4:     $S_p \leftarrow S_p \setminus g$
5:     if DSS$(\mathcal{P}) >$ COUNT$(g)$ then
6:         CONTINUE
7:     $\mathcal{P} \leftarrow$ PatternMaintain$(\mathcal{P}, g, D)$
8:     $\mathcal{P}_g \leftarrow$ RightMostExtend$(g, D, E_{max})$
9:     for $g' \in \mathcal{P}_g$ do
10:        if PRM$(g, g') < (1+\alpha)$Score$_L + \frac{(1-\alpha)|Cov(\mathcal{P},D)|}{k}$ then
11:            $\mathcal{P}_g \leftarrow \mathcal{P}_g \setminus g'$
12:    $S_p \leftarrow \mathcal{P} \cup \mathcal{P}_g$
```

THEOREM 4. *Let $\mathcal{P}_{opt}$ be an optimal solution to top-k edge-diversified patterns discovery problem. The approximation ratio of patterns $\mathcal{P}$ generated by TED (and TED_BASE) is bounded by $\frac{|Cov(\mathcal{P},D)|}{|Cov(\mathcal{P}_{opt},D)|} \geq \frac{1}{4}$.*

THEOREM 5. *Worst case time and space complexities of TED are $O(|D|2^{max(V(G))^2})$ and $O(max(E(G))|D|)$, where $max(V(G))$ (resp. $max(E(G))$) is maximum number of vertices (resp. edges) in graph $G \in D$, and $|D|$ is number of graphs in D.*

6 TEDLITE

Both subgraph enumeration on an entire database and top-k patterns search over all enumerated subgraphs can be computationally expensive for an even larger graph database D. To alleviate this problem, we develop a lightweight TED (denoted by TEDLITE). Intuitively, TEDLITE is to sample a subset of graphs D' from D and then perform TED algorithm on D'. The questions become: how to generate sample graphs and how many graphs should be sampled.

To answer the first question, TEDLITE adopts a random sampling method to sample $|D'|$ graphs from D. Intuitively, the sampled graphs D' are supposed to approximately keep graph property distribution (*e.g.,* label distribution) of original database D. To this end, it first represents each graph as a feature vector, where each entry indicates the frequency of a particular label. Then, graphs are grouped into several clusters by utilizing k-means clustering algorithm. As a result, graphs in each cluster are supposed to have similar label distribution. Therefore, TEDLITE randomly select $|C'_i|$ graphs (denoted by C'_i) from a cluster C_i where $|C'_i|$ should be directly proportional to the size of C_i, *i.e.,* $|C'_i| = |C_i||D'|/|D|$.

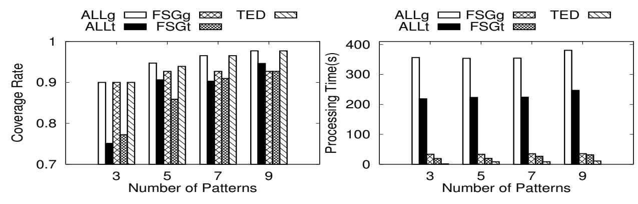
For the second question, the intuition of TEDLITE is to ensure that the absolute difference of edge frequency in D' and D are theoretically guaranteed. Formally, given the average edge number of graphs (denoted by $|E_{avg}|$) and an edge e, the absolute difference of its frequency in D' and D is $Dif_{abs} = |fr(e, D) - fr(e, D')|$ where $fr(\cdot)$ is a frequency. The following Theorem 6 shows how to determine a sample size $|D'|$.

THEOREM 6. *Given an error bound ϵ and a maximum probability ρ for the absolute difference that exceeds ϵ, the size of the random sample (*i.e., *$|D'|$) is determined by $|D'| \geq \frac{1}{2\epsilon^2|E_{avg}|} ln\frac{2}{\rho}$.*

According to Theorem 6, the size of the random samples is determined by an error bound ϵ and a maximum probability ρ for the absolute difference that exceeds ϵ. For example, given D and sampling parameters $\rho = 0.01$ and $\epsilon = 0.02$, if $|E_{avg}| = 40$, D' should consist of at least $\frac{1}{2(0.02)^2 \times 40} ln\frac{2}{0.01} = 166$ random samples. As we shall see in Section 7, TEDLITE achieves a good balance between the quality of discovered patterns (*i.e.,* coverage rate) and the runtime performance (*i.e.,* processing time).

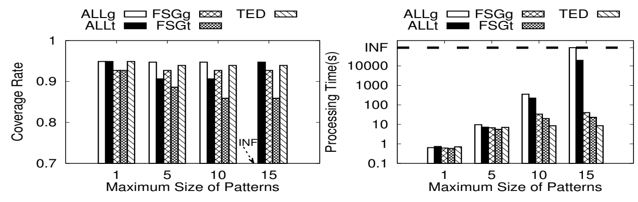
7 PERFORMANCE STUDY

In this section, we investigate the performance of TED and report the key findings. TED is implemented with Java (JDK1.8). All experiments are conducted on a 64-bit Windows desktop with AMD Ryzen 5 3500X 6-Core CPU (3.6GHz) and 32GB of main memory.



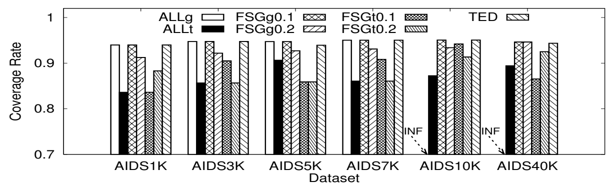
(a) Coverage Rate **(b) Processing Time**

Figure 7: Effect of Number of Patterns.

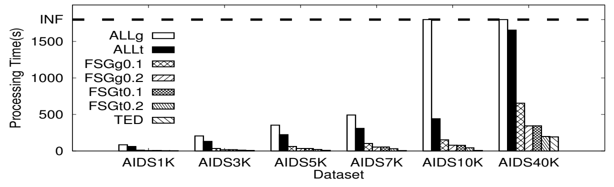


(a) Coverage Rate **(b) Processing Time**

Figure 8: Effect of Maximum Size of Patterns.

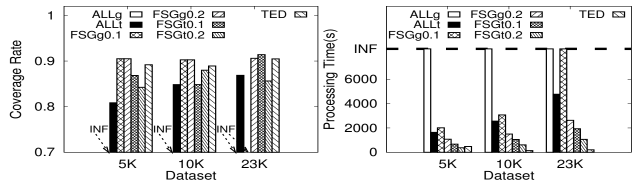


(a) Coverage Rate



(b) Processing Time

Figure 9: Baseline comparison on Aids.



(a) Coverage Rate **(b) Processing Time**

Figure 10: Baseline comparison on PubChem.

7.1 Experimental Setup

Datasets. The experiments are conducted on three datasets. (1) The dataset AIDS antiviral [6] consists of 40,000 (40K) data graphs. (2) The dataset *PubChem* [7] consists of many chemical compound graphs. Unless otherwise stated, *PubChem* refers to the 23K dataset. Other variants used are 100K, 300K, 500K and 1 million (1M). (3) The dataset *eMol* [8] consists of 10K chemical compounds. Note that <Y><X> are used to denote variants of various datasets, where *Y* and *X* refer to the name of the dataset and the number of graphs used, respectively. For example, AIDS10K refers to the dataset AIDS consisting of 10K data graphs.

Baselines. We compare Ted against the proposed baselines and their variants: (1) enumerating all subgraphs and then performing greedy search (ALL$_g$, Algorithm 1), (2) integrating enumeration of all subgraphs and swapping-based search (ALL$_t$, *i.e.,* a variant of ALL$_g$), (3) enumerating all frequent subgraphs and then performing greedy search (FSG$_g$, Algorithm 2), and (4) integrating enumeration of all frequent subgraphs and swapping-based search (FSG$_t$, *i.e.,* a variant of FSG$_g$). We also compare Ted with its variants (BASE, DSS and PRM) where (1) BASE: Ted_base (Algorithm 3), (2) DSS: BASE + Ted_dss (Algorithm 5), (3) PRM: DSS + Ted_prm (Algorithm 6) and (4) TED (*i.e.,* Algorithm 8): PRM + Ted_ips (Algorithm 7).

Parameter settings. Unless specified otherwise, we set $k = 5$ and $E_{max} = 10$.

Performance measures. We use the following measures for performance evaluation: (1) *Processing Time (in second):* time taken to generate the patterns, (2) *Coverage Rate:* the coverage rate of patterns to the total number of edges in a database *D*, (3) *Index Time (in second):* time taken to maintain PES-index, (4) *Index Size (in kilobyte):* space consumption for storing the PES-index. Note that INF is used to denote the case that a test does not stop in a time limit (24h) or incurs memory exception.

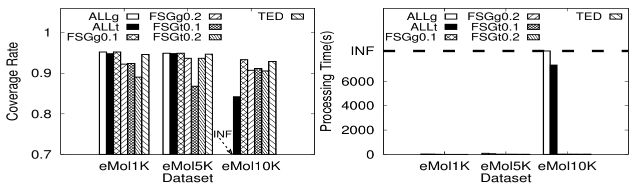
[6]https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data
[7]ftp://ftp.ncbi.nlm.nih.gov/pubchem/Compound/CURRENT-Full/SDF/
[8]https://www.emolecules.com/info/plus/download-database

7.2 Experimental Results

Exp 1: Setting of Maximum Size and Number of Patterns. We first vary the number of patterns (*i.e., k*) on AIDS5K. Figure 7 plots the results. In general, the coverage rate and processing time increase with *k*, since more patterns are introduced and higher coverage will be obtained as *k* increases. The methods based on greedy search (*i.e.,* ALL$_g$ and FSG$_g$) generally show better coverage rate and more processing time compared to swapping-based search methods (*i.e.,* ALL$_t$ and FSG$_t$), as the former needs to store all (resp. frequent) subgraphs for further searching. However, TED is always comparable to ALL$_g$ and better than other methods in terms of coverage rate (Figure 7(a)), although it is based on swapping-based search. As shown in Figure 7(b), TED requires less processing time than other methods on all settings. In the following experiments, we set $k = 5$.

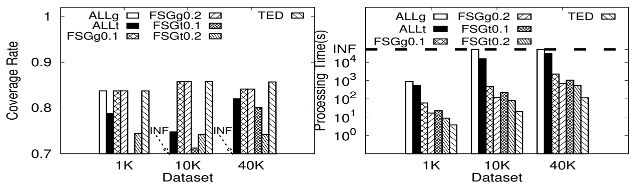
We also study the performance of Ted with different maximum sizes of patterns (E_{max}) on AIDS5K. As shown in Figure 8, TED is comparable to ALL$_g$ and better than other methods in terms of coverage rate, but requires less processing time in most cases, especially for larger E_{max}. Intuitively, ALL$_g$ can obtain better coverage rate compared to other methods except TED since all subgraphs are enumerated and stored, which makes it memory- and time-consuming (*e.g.,* INF for $E_{max} = 15$). In addition, as E_{max} increases, search space is enlarged and hence processing time of these methods increases. Note that coverage rate fluctuates within a narrow range, which is due to the label distribution of edges in the database. We set $E_{max} = 10$ in the following experiments unless specified.

Exp 2: Comparison with Baselines. Next, we compare Ted with baselines and their variants, *i.e.,* ALL$_g$, ALL$_t$, FSG$_g$, and FSG$_t$



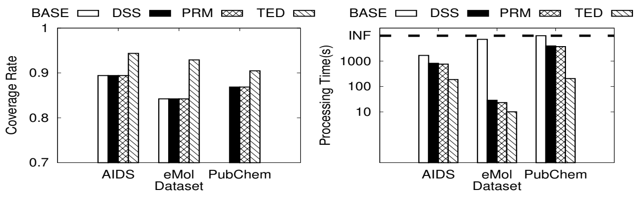
(a) Coverage Rate **(b) Processing Time**

Figure 11: Baseline comparison on eMol.



(a) Coverage Rate **(b) Processing Time**

Figure 12: Baseline comparison on AidsL.



(a) Coverage Rate **(b) Processing Time**

Figure 13: Effect of Optimization Strategies.

on AIDS dataset in Figure 9. The results of FSG-based algorithms (i.e., FSG_g and FSG_t) with various support (0.1 and 0.2) are also reported. As depicted in Figure 9(a), in terms of coverage rate, Ted outperforms other baselines and is comparable to ALL_g, which incurs INF on AIDS10K and AIDS40K. As the size of dataset increases from 1K to 40K, processing time of ALL_g dramatically increases while that of our TED algorithm steadily increases to less than 4 minutes on AIDS40K, as shown in Figure 9(b).

We further compare Ted with baselines and their variants on another two datasets, *PubChem* and *eMol*. As depicted in Figure 10(a) and Figure 11(a), although FSG_g algorithm often shows comparable coverage rate to TED, its processing time dramatically increases with the size of dataset and even incurs INF (*e.g.,* on PubChem23K), as it adopts greedy search on all frequent subgraphs whose size is data-dependent (Figure 10(b) and Figure 11(b)). In addition, FSG_g can not theoretically guarantee the quality of final patterns. Note that ALL_g incurs a lot of INF exceptions on *PubChem* dataset since graphs in the dataset are relatively larger in size (average and maximum number of edges are 44 and 838, respectively).

To sum up, Ted outperforms baselines in terms of both coverage rate and processing time.

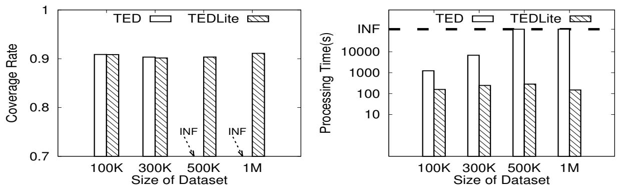
Exp 3: Effect of Optimization Strategies. We further evaluate the effectiveness of optimization strategies by comparing BASE, DSS, and PRM with TED where all optimization strategies are used. The results are presented in Figure 13. Comparing to BASE, DSS, and PRM, TED reduces the processing time but does not decrease the coverage rate, since it eliminates unpromising patterns but without sacrificing promising ones. Thanks to all adopted optimization strategies, TED is the best one in terms of coverage rate and processing time compared to BASE, DSS, and PRM. In addition, we can observe that processing time of BASE, DSS, PRM, and TED shows a decreasing trend, which further justifies the effectiveness of optimization strategies. Similar observations are made on AIDS10K-40K graphs (details refer to [1]).

Table 2: Size of PES-Index

Dataset	Aids		eMol		PubChem	
	$10K$	$40K$	$5K$	$10K$	$10K$	$23K$
Index Size(KB)	234	1008	89	157	428	1157
Index/Graphs (%)	5.39	5.31	5.40	5.39	5.80	7.58

Table 3: Maintenance Time of PES-Index

Dataset	Aids		eMol		PubChem	
	$10K$	$40K$	$5K$	$10K$	$10K$	$23K$
Index Time(s)	0.5	1.88	0.25	0.37	1.1	2.85
Index Time/Total (%)	6.86	1.00	4.12	3.63	0.78	1.39



(a) Coverage Rate **(b) Processing Time**

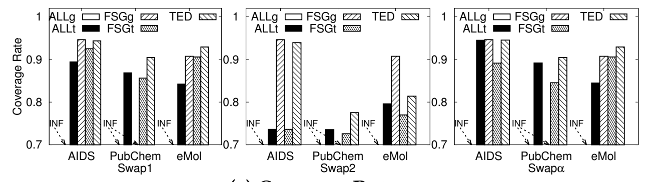
Figure 14: Scalability Test.

Exp 4: PES-Index Test. In this experiment, we test the size and maintenance time of PES-Index. As presented in Table 2, PES-Index size increases with the size of the dataset. Note that in comparison with the size of dataset, PES-Index size is very small since only five components are stored in PES-Index. In particular, for the larger datasets AIDS40K and PubcChem23K, the space taken to store PES-Index is only 5.31% and 7.58% of the size of the underlying dataset.

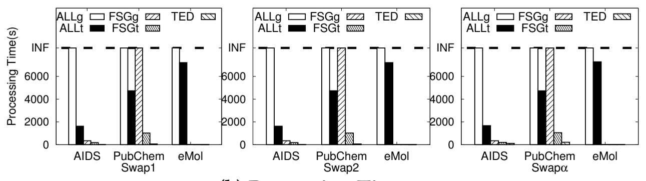
We also report the maintenance time of PES-Index in Table 3. Observe that as the size of dataset increases, maintenance time increases accordingly but makes up less than 7% of the total processing time. For example, maintenance time of PES-Index on AIDS40K and PubcChem23K are 1% and 1.39% of the corresponding total processing time, respectively.

Exp 5: Scalability Test. Figure 14 reports the performance of Ted and TedLite on very large dataset of PubChem. We observe that both of them can efficiently handle large datasets. In particular, Ted is able to process up to 300K graphs within about 1.9 hours and achieves high coverage rate (*i.e.,* more than 90%). Compared to Ted, TedLite has slightly worse results in terms of coverage rate but takes far less processing time (e.g., 150s for 1M graphs in PubChem).

Exp 6: Effect of Swapping Criteria. We investigate the effect of different swapping criteria (see Section 4), namely, $Swap_1$, $Swap_2$



(a) Coverage Rate



(b) Processing Time

Figure 15: Effect of Swapping Criteria.

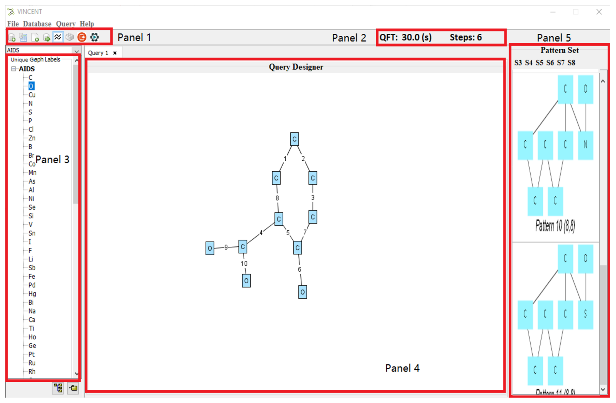


Figure 16: Interface of VINCENT.

and $Swap_\alpha$. The results are reported in Figure 15. In general, TED outperforms baselines in terms of both coverage rate and processing time no matter what swapping criteria is used. Although FSG$_g$ with $Swap_2$ shows higher coverage rate on EMOL, as discussed in Exp 2, the results generated by FSG$_g$ are data-dependent and not theoretically guaranteed. Furthermore, it may incur INF on larger datasets (*e.g.*, PUBCHEM). This experiment further justifies the effectiveness of TED algorithm.

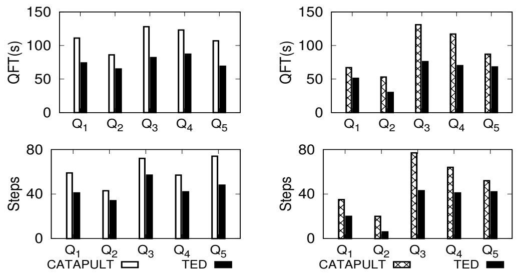


Figure 17: Results of User Study.

Exp 7: User Study. The most pertinent question related to TED is *whether top-k edge-diversified patterns can facilitate existing or potential applications?* To answer the question, we develop a demonstration system called VINCENT, as shown in Figure 16, based on which user studies are conducted. We recruit 15 unpaid volunteers (ages from 20 to 32) in accordance to HCI research that recommends at least 10 participants [27, 28]. These volunteers are students or researchers from different majors. All of them are pre-trained to use VINCENT before conducting the user studies.

The first study is to ask these volunteers to visually formulate queries with VINCENT. For each dataset AIDS or PUBCHEM, we first select 5 queries of size in the range [10-40]. Then, the generated top-k edge-diversified patterns are displayed on the Panel 5 in Figure 16. Volunteers are allowed to visually formulate the queries by dragging and dropping patterns from Panel 5 and nodes from Panel 3 to Panel 4 to formulate queries. The *Query Formulation Time* (QFT) and *Steps* taken are recorded in Panel 2. Finally, we compare the results with *canned patterns* generated by state-of-the-art called CATAPULT [17], which towards efficient visual query formulation. Figure 17 reports the results on PUBCHEM (left) and AIDS (right). Observe that compared to CATAPULT, TED facilitates more efficient (shorter QFT and lesser number of steps) query formulation.

User study on exploratory subgraph search is also conducted. Volunteers are allowed to process subgraph query, and explore the query results with TED EXPLORER (details refer to [1]) where users can customize the pattern size. Observe that 20% of time taken for exploratory search are saved.

The most appealing thing to us is that top-k edge-diversified patterns generated by TED contain not only patterns with statistical significance (*e.g.*, frequent patterns) but also infrequent patterns with biological importance. For example, Pattern 10 displayed on the Panel 5 in Figure 16 is an important organic compound *Nitrosobenzene*, which is one of the prototypical organic nitroso compounds. Therefore, TED may open up new opportunities to bioinformatics, drug design, etc.

8 RELATED WORK

Subgraph enumeration listing all subgraphs or counting all instances of a particular graph in a graph database has been extensively studied in the literature. Many of these approaches are based on dynamic programming and techniques such as color coding. [3] develops a simple edge-searching method for listing all triangles, quadrangles, and complete subgraphs. But this method may incur significant disk reads when it is applied to external subgraph enumeration. Compared to [3], [5] presents an enhanced algorithm based on symmetry-breaking technique. In addition, [2] applies the color coding technique and dynamic programming routine for counting non-induced occurrences of subgraph topologies in the form of trees and subgraphs. A parallel color coding and streaming algorithm for subgraph enumeration in large social contact networks is also designed [6]. To handle the problem that data graphs cannot reside chronically in memory, a disk-based algorithm DUALSIM [32] is proposed. Since subgraph enumeration is computationally expensive, distributed approaches have been proposed. [29] explores a technique to process subgraph enumeration in a single map-reduce round. Another novel parallel framework named PSgL is built upon Giraph [31]. In addition, Lai et al. develop TwinTwig [30] on MapReduce. Other parallel algorithms [33, 34] for compressing the intermediate results are also considered. Instead of enumerating subgraphs with CPU, GPU-based subgraph enumeration methods [35–37] are also explored. Our top-k edge-diversified patterns discovery problem is orthogonal to these work as subgraph enumeration is a necessary step in our TED framework.

Instead of enumerating all subgraphs, frequent subgraph mining (FSM) is to generate only frequent subgraphs. Existing FSM methods consider two settings, transactional and single graph such as [13]. The transactional case assumes the graph database contains a set of graphs, which is the same as our setting in this work. Both AGM [7] and FSG [8] methods adopt BFS-based strategy to generate frequent subgraphs. They first generate a lot of candidates and then perform subgraph isomorphism test to prune false positives. These methods are time-consuming since subgraph isomorphism test is NP-complete and costly. To address this problem, Yan et al. propose gSpan [10] algorithm, which is the first DFS-based method for FSM. Owing to its better performance, it is widely used. FFSM [11] claiming to be competitive with gSpan [10] is the method for efficiently handling the underlying subgraph isomorphism problem in AGM and FSG. Other methods such as [12] adopt the pattern growth strategy. Other than enumerating all frequent subgraphs, MARGIN [38] and CloseGraph [39] generate maximal frequent subgraphs and closed frequent subgraphs, respectively. However, frequent subgraphs may be topologically similar to each other, and users involving in real-world applications are not only interested in frequent subgraphs. Although there are methods [17, 40, 41] which aim to discover subgraphs that are not necessarily frequent, none of them focus on top-k edge-diversified problems and their solutions can not be directly adapted for it.

Diversity problem is the most germane to this research, which seeks for search results with diversity. [42] presents diversity-aware search method of relevant documents. Fan et al. [14] aims to retrieve diversified top-k matches for a given vertex. The problem of finding redundancy-aware maximal cliques is considered by [43]. In addition, [44] further studies the diversified top-k clique search problem, which is to find k maximal cliques in a data graph so that the maximum number of vertices are included. Given a query graph, [45] retrieves diversified top-k matches to cover more vertices. Other than static graph, diversified pattern mining on temporal network is also studied by [46], which is for finding k diversified *dense temporal subgraphs*. Recent work [47] investigates the problem of diversity-based shortest path. To the best of our knowledge, our work is the first one to study top-k edge-diversified patterns discovery problem.

9 CONCLUSION

In this work, we investigate the top-k edge-diversified patterns discovery problem, which is to find k subgraphs from a graph database such that the maximum number of edges in the database are covered. We present a novel framework called TED and an efficient index structure for this problem. In addition, three optimization strategies are developed to improve the performance. To handle even larger graph databases, a lightweight version called TEDLITE is further designed. TED requires limited memory and achieves a guaranteed approximation ratio. Extensive experimental results justify the superiority of TED over baseline solutions.

As for future work, we plan to study the problem of discovering top-k edge-diversified patterns over other graph databases such as a single large graph (*e.g.,* social network), weighted graphs and directed graphs.

REFERENCES

[1] Technical Report. Available at: https://github.com/TechReport2022/TED/blob/main/ICDE-TED-TR.pdf.
[2] Alon N, Dao P, Hajirasouliha I, et al. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13): i241-i249, 2008.
[3] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1), 1985.
[4] Gonen M, Ron D, Shavitt Y. Counting stars and other small subgraphs in sublinear time. *In SODA*, 2010.
[5] Grochow J A, Kellis M. Network motif discovery using subgraph enumeration and symmetry-breaking. *In RECOMB*, 2007.
[6] Zhao Z, Khan M, Kumar V S A, et al. Subgraph enumeration in large social contact networks using parallel color coding and streaming. *In ICPP*, 2010.
[7] Inokuchi A, Washio T, Motoda H. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data, *In ECML-PKDD*, 2000.
[8] Kuramochi M, Karypis G. An Efficient Algorithm for Discovering Frequent Subgraphs. *TKDE*, 16(9):1038-1051, 2004.
[9] Gudes E, Shimony S E, Vanetik N. Discovering Frequent Graph Patterns using Disjoint Paths. *TKDE*, 18(11): 1441-1456, 2006.
[10] Yan X, Han J W. gSpan: Graph-based Substructure pattern mining. *In ICDM*, 2002.
[11] Huan J, Wang W, Prins J. Efficient Mining of Frequent Subgraph in the Presence of Isomorphism. *In ICDM*, 2003.
[12] Nijssen S, Kok J N. A Quickstart in Frequent Structure Mining can Make a Difference. *In KDD*, 2004.
[13] Elseidy M, Abdelhamid E, Skiadopoulos S, et al. Grami: Frequent subgraph and pattern mining in a single large graph. *In VLDB*, 2014.
[14] Fan W, Wang X, Wu Y. Diversified top-k graph pattern matching. *PVLDB*, 6(13): 1510-1521, 2013.
[15] Gollapudi S, Sharma A. An axiomatic approach for result diversification. *In WWW*, 2009.
[16] Qin L, Yu J X, Chang L. Diversifying top-k results. *PVLDB*, 5(11), 2012.
[17] Huang K, Chua H E, Bhowmick S S, et al. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. *In SIGMOD*, 2019.
[18] Bhowmick S S, Huang K, Chua H E, et al. AURORA: Data-driven Construction of Visual Graph Query Interfaces for Graph Databases. *In SIGMOD*, 2020.
[19] Huang K, Bhowmick S S, Zhou S, et al. Picasso: exploratory search of connected subgraph substructures in graph databases. *PVLDB*, 10(12): 1861-1864, 2017.
[20] White R W, Roth R A. Exploratory search: Beyond the query-response paradigm. *Synthesis lectures on information concepts, retrieval, and services*, 1(1): 1-98, 2009
[21] Chua H E, Bhowmick S S, Tucker-Kellogg L. Synergistic target combination prediction from curated signaling networks: Machine learning meets systems biology and pharmacology. *Methods*, 129:60-80, 2017.
[22] Feige U. A threshold of ln n for approximating set cover. *Journal of ACM*, 45(4): 634-652, 1998.
[23] Saha B, Getoor L. On maximum coverage in the streaming model & application to multi-topic blog-watch. *In SDM*, 2009.
[24] Ausiello G, Boria N, Giannakos A, et al. Online maximum k-coverage. *In FCT*, 2011.
[25] Yuan D, Mitra P, Yu H, Giles C L. Updating graph indices with a one-pass algorithm. *In SIGMOD*, 2015.
[26] Toivonen H. Sampling large databases for association rules. *In VLDB*, 1996.
[27] Lazar J, Feng J H, Hochheiser H. Research methods in human-computer interaction. *John Wiley & Sons*, 2010.
[28] Faulkner, Laura. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 2003.
[29] Afrati F N, Fotakis D, and Ullman J D. Enumerating subgraph instances using map-reduce. *In ICDE*, 2013.
[30] Lai L, Qin L, Lin X, et al. Scalable subgraph enumeration in mapreduce. *VLDBJ*, 8(10): 974-985, 2015.
[31] Shao Y, Cui B, Chen L, et al. Parallel subgraph listing in a large-scale graph. *In SIGMOD*, 2014.
[32] Kim H, Lee J, Bhowmick S S, et al. Dualsim: Parallel subgraph enumeration in a massive graph on a single machine. *In SIGMOD*, 2016.
[33] Sun S, Che Y, Wang L, et al. Efficient parallel subgraph enumeration on a single machine. *In ICDE*, 2019.
[34] Qiao M, Zhang H, Cheng H. Subgraph matching: on compression and computation. *In PVLDB*, 2017.
[35] Lin W, Xiao X, Xie X, and Li X L. Network motif discovery: A gpu approach. *In ICDE*, 2015.
[36] Tran H N, Kim J J, and He B H. Fast subgraph matching on large graphs using graphics processors. *In DASFAA*, 2015.
[37] Guo W, Li Y, Sha M, et al. GPU-accelerated subgraph enumeration on partitioned graphs. *In SIGMOD*, 2020.
[38] Thomas L T, Valluri S R, and Karlapalem K. MARGIN: Maximal frequent subgraph mining. *In TKDD*, 2010.
[39] Yan X and Han J. CloseGraph: mining closed frequent graph patterns. *In SIGKDD*, 2003.
[40] Yan X, Cheng H, Han J, and Yu P S. Mining significant graph patterns by leap search. *In SIGMOD*, 2008.
[41] Ranu S and Singh A K. GRAPHSIG: A scalable approach to mining significant subgraphs in large graph databases. *In ICDE*, 2009.
[42] Angel A. and Koudas N. Efficient diversity-aware search. *In SIGMOD*, 2011.
[43] Wang J, Cheng J, and Fu A. Redundancy-aware maximal cliques. *In KDD*, 2013.
[44] Yuan L, Qin L, Lin X, Chang L, and Zhang W. Diversified top-k clique search. *In ICDE*, 2015.
[45] Yang Z, Fu A, Liu R. Diversified top-k subgraph querying in a large graph. *In SIGMOD*, 2016.
[46] Yang Y, Yan D, Wu H, et al. Diversified temporal subgraph pattern mining. *In SIGKDD*, 2016.
[47] Chondrogiannis, Theodoros, et al. Finding k-shortest paths with limited overlap. *VLDBJ*, 29(5): 1023-1047, 2020.