Top-k Graph Summarization on Hierarchical DAGs

Xuliang Zhu, Xin Huang, Byron Choi, Jianliang Xu
Hong Kong Baptist University, Hong Kong, China
{csxlzhu,xinhuang,bchoi,xujl}@comp.hkbu.edu.hk

ABSTRACT

Directed acyclic graph (DAG) is an essentially important model to represent terminologies and their hierarchical relationships, such as Disease Ontology. Due to massive terminologies and complex structures in a large DAG, it is challenging to summarize the whole hierarchical DAG. In this paper, we study a new problem of finding k representative vertices to summarize a hierarchical DAG. To depict diverse summarization and important vertices, we design a summary score function for capturing vertices' diversity coverage and structure correlation. The studied problem is theoretically proven to be NP-hard. To efficiently tackle it, we propose a greedy algorithm with an approximation guarantee, which iteratively adds vertices with the large summary contributions into answers. To further improve answer quality, we propose a subtree extraction based method, which is proven to guarantee achieving higher-quality answers. In addition, we develop a scalable algorithm k-PCGS based on candidate pruning and DAG compression for large-scale hierarchical DAGs. Extensive experiments on large real-world datasets demonstrate both the effectiveness and efficiency of proposed algorithms.

CCS CONCEPTS

• **Information systems** → **Hierarchical data models**; **Summarization**; • **Theory of computation** → **Graph algorithms analysis**.

KEYWORDS

Directed acyclic graph; Data summarization; Hierarchy

ACM Reference Format:
Xuliang Zhu, Xin Huang, Byron Choi, Jianliang Xu. 2020. Top-k Graph Summarization on Hierarchical DAGs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411899

1 INTRODUCTION

Directed acyclic graph is an essential model widely adopted to represent hierarchical terminologies and their relation structure, such as Disease Ontology [1, 15], Gene Ontology [14], Image-net [7], Medical Entity Directory [16], ACM computing classification system [25], and so on. In many real-world applications of DAGs, vertices have not only edge relationships but also *node weights*. For instance, in biomedicine, the weight of a terminology can be the aggregate count of events, such as the occurrences of diseases [15].

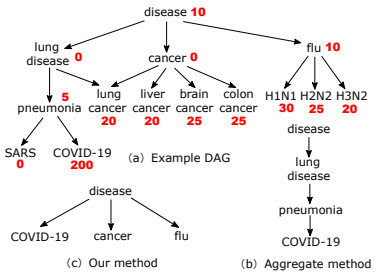


Figure 1: A disease ontology for DAG summarization.

However, the massive terminologies and complex structures bring significant challenges to understanding a DAG dataset. Graph summarization could give a direct and human-friendly overview of the dataset being analyzed. Unfortunately, graph summarization works only when the complexity of summarized results is within human cognitive capacity; otherwise, the massive size of terminologies may overwhelm any user who tries to understand or visualize it [15, 24]. For example, it is impossible to comprehend the whole SNOMED CT dataset [2], which contains more than 311 thousand medical concepts. Hence, it is important to reduce a large DAG dataset to a manageable size using summarization. Furthermore, hierarchical DAGs summarization has a wide range of applications such as summarized recommendation, visual data exploration [4], and snippet generation for information search [8, 9].

Motivating Example. We show a motivating example of DAG summarization studied in this paper. Figure 1(a) shows an example of hierarchical disease ontology in DAG, where each vertex represents a disease terminology. The associated weight of each vertex indicates the occurrence of the disease. A directed edge from one vertex to another vertex represents the concept-instance relationship, e.g., "pneumonia" is a general concept of two instances "SARS" and "COVID-19". Assume that this DAG represents the disease occurrences of hospitals in a city. It contains 14 diseases with different occurrence weights. The task is to find a small number $k = 4$ representative diseases to summarize this DAG. We consider four different methods in the following.

- **Top-k largest weights**. One simple method is to pick four vertices with the largest weights, i.e., { "COVID-19", "H1N1", "H2N2", "colon cancer"}. However, these vertices are instances with very limited representation to generalize other vertices, such as "H3N2", "brain cancer", and so on.
- **Aggregate method [15].** Jing and Cimino [15] proposed a statistical approach to pick a set of nodes with the highest aggregate weights, i.e., { "disease", "lung disease", "pneumonia", "COVID-19"}, as shown in Figure 1(b). However, the approach suffers

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411899

from the significant limitation of lacking diversity. All of "disease", "lung disease", and "pneumonia" are general concepts of the disease "COVID-19", which has the largest weight of 200. It misses other important diseases, e.g., cancer and flu. Thus, the summarization diversity is low.

- **GDVO [14].** GDVO finds a set of representative vertices to summarize a tree, which can be regarded as a special case of DAG. However, the data structure of a tree has a limited ability to represent complex relations, such as the relations of "disease → cancer → lung cancer" and "disease → lung disease → lung cancer" in Figure 1(a).
- **Our approach.** Figure 1(c) shows our expected solution, which offers a better summarized view: "disease" is a general concept representing all vertices in general; "cancer" and "flu" represent two categories of multiple diseases with large weights; "COVID-19" is the most important disease with the largest weight, which has a more detailed summarization than "pneumonia" as no "SARS" disease happened in this DAG.

Motivated by this example, we study the problem of top-k graph summarization for large hierarchical DAGs. Specifically, given a DAG with node weights, the problem is finding a small set of k representative vertices to summarize the whole DAG. To model the summary impact of a vertex, we design a summary score function for capturing the diversity coverage and structure correlation. We also show that the problem is NP-hard and challenging for developing fast algorithms. To tackle it, we first propose a greedy strategy based method Greedy+. The algorithm finds an approximate answer by iteratively adding vertices with the largest contribution for summary score, until the answer has k representative vertices.

Based on Greedy+, we develop two improved methods to trade off the answer quality and algorithm efficiency. Specifically, we propose an effective EXT-Greedy algorithm for improving the answer quality. EXT-Greedy extracts a subtree from DAG based on the top-k answer vertices by Greedy+ and applies tree-based dynamic programming techniques to obtain a better result. We show that this method consistently outperforms Greedy+ in terms of quality, through both theoretical analysis and experimental evaluation. In addition, to scale up with large-scale hierarchical DAGs, we propose an efficient algorithm k-PCGS, which can achieve the same result as Greedy+ in a faster way. k-PCGS equips with two key techniques of candidate pruning and DAG compression, which can prune lots of disqualified candidates and reduce the DAG size.

To summarize, this paper makes the following contributions:

- We motivate the problem of graph summarization for large hierarchical DAGs. Specifically, we propose a summary score function satisfying the good desiderata of diversity coverage and structure correlation for important vertices. Based on this, we formulate our problem as finding a set of k representative vertices with the largest summary score to summarize a DAG, called the kDAG-problem (Section 2).
- We analyze the hardness of our problem and formally prove it as NP-hard. We also discuss the properties of our summary score function and the alternative choices of representative correlation functions (Section 3).
- We propose a greedy algorithmic framework to tackle our problem. It iteratively selects a representative vertex with the largest summary score contribution into the answer. The greedy strategy is effective with a $(1 - \frac{1}{e})$-approximation guarantee. (Section 4).
- We propose an effective EXT-Greedy algorithm based on the subtree extraction and tree-based summarization. EXT-Greedy is proven to guarantee high-quality answers no worse than Greedy+. (Section 5).
- We design several useful upper bounds and lower bounds to estimate the summary contribution of a vertex. Based on the bounds, we propose a pruning theorem to eliminate disqualified candidates for saving computations. Moreover, we develop a graph specification technique to compress DAG by discarding tree-shape structures and useless vertices. Integrating with candidate pruning and DAG compression, we propose a fast algorithm k-PCGS, which can achieve the same answer as Greedy+ on large real-world DAGs. (Section 6).
- We conduct extensive experiments on real-world datasets to validate the efficiency and effectiveness of our proposed algorithms for DAGs and trees (Section 7).

We discuss related work in Section 8 and conclude the paper in Section 9.

2 PRELIMINARIES

In this section, we define the notions and our problem.

2.1 Directed Acyclic Graph

Let $G = (V, E, \mathsf{feq})$ be a directed acyclic graph (DAG) with a set V of vertices, a set E of edges, and a weight function of vertices as $\mathsf{feq} : V \to \mathbb{R}^{\geq 0}$. The edges of G have directions, but G has no directed cycles. Let $n = |V|$ and $m = |E|$ be the numbers of vertices and edges, respectively. W.l.o.g. we assume that $m \geq n - 1$ and $n \in O(m)$, following [19]. For an edge $\langle u, v\rangle$, we say u is an in-neighbor of v and v is an out-neighbor of u. For a vertex v in G, we denote the set of in-neighbors of v by $\mathsf{N}^+(v) = \{u : \langle u, v\rangle \in E\}$ and the set of out-neighbors of v by $\mathsf{N}^-(v) = \{u : \langle v, u\rangle \in E\}$. The weight of vertex v is denoted by $\mathsf{feq}(v) \in \mathbb{R}^{\geq 0}$. The larger is the weight of a node, the more is its importance.

For any two vertices u and v, we say that u can reach v (denoted as $u \to v$), if and only if there exists a directed path from u to v in G. The connectivity in G is always weak, as there exists no directed cycles in a DAG for strong connectivity. If neither $u \to v$ nor $v \to u$ holds, u and v are disconnected. Note that $v \to v$. The longest distance between $u \to v$ in G is called the height of G, denoted as h. In the following, we introduce two useful concepts of ancestors and descendants.

Definition 1 (Ancestors). *The ancestors of a vertex v, denoted by* $\mathsf{anc}(v)$*, are the set of vertices that can reach v in G, i.e.,* $\mathsf{anc}(v) = \{u \in V : u \to v\}$.

Definition 2 (Descendants). *The descendants of a vertex v, denoted by* $\mathsf{des}(v)$*, are the set of vertices that are reachable from v in G, i.e.,* $\mathsf{des}(v) = \{u \in V : v \to u\}$.

Example 1. *Figure 2 shows an example of* DAG*. The vertex v_2 has the in-neighbor* $\mathsf{N}^+(v_2) = \{v_1\}$*, out-neighbors* $\mathsf{N}^-(v_2) = \{v_3, v_4, v_5\}$*, ancestors* $\mathsf{anc}(v_2) = \{v_1, v_2\}$*, and descendants* $\mathsf{des}(v_2) = \{v_2, v_3, v_4, v_5, v_6\}$.

2.2 Representative Correlation

Before formally defining the representative correlation, we introduce the directed distance. In a DAG of concept hierarchy, two

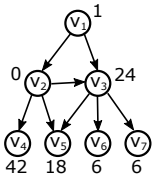


Figure 2: The DAG used in the running example

concept entities u and v are regarded to be correlative if there exists a directed path between u and v. Specifically, we say u is a *general* concept of v if u is an ancestor of v, i.e., $v \rightarrow u$. In contrast, v is an *instance* concept of u if v is a descendant of u, i.e., $u \rightarrow v$. Obviously, the strength of a correlation between u and v is reflected by their distance. The smaller is the distance, the closer is the correlation. We represent the distance from u to v, denoted by $\text{dist}\langle u, v\rangle$, as the length of the shortest directed path from u to v in G. Note that $\text{dist}\langle v, v\rangle = 0$, and $\text{dist}\langle u, v\rangle = +\infty$ if $v \nrightarrow u$. Based on the distance, we define the correlation as follows.

Definition 3 (Correlation). *Given two vertices v, u, the representative correlation of v to u is* $\text{cor}\langle v, u\rangle = \frac{1}{1+\text{dist}\langle v,u\rangle}$ *where* $\text{dist}\langle v, u\rangle$ *is the shorest distance from v to u.*

The smaller is $\text{dist}\langle v, u\rangle$, the higher is $\text{cor}\langle v, u\rangle$. If $u \notin \text{des}(v)$, $\text{cor}\langle v, u\rangle = 0$, indicating that the correlation impact of v for u is zero. For example, in the figure 1(a), "lung cancer" is not a good general concept of "COVID-19", as they are irrelevant to each other. On the other hand, "pneumonia" could be a good general concept of "COVID-19", and even better than "lung disease" and "disease". Definition 3 implies the key property that should be obeyed by a good score function of correlation. There exist many other choices of the $\text{cor}\langle v, u\rangle$ function. We further discuss different correlation functions and analyze their properties in Section 3.2.

Example 2. *In the* DAG *G shown in Figure 2, there exists one shortest path from v_2 to v_5 as $\{\langle v_2, v_5\rangle\}$ and* $\text{dist}\langle v_2, v_5\rangle = 1$. *Moreover,* $\text{dist}\langle v_2, v_2\rangle = 0$ *and* $\text{dist}\langle v_3, v_4\rangle = +\infty$. *Correspondingly,* $\text{cor}\langle v_2, v_5\rangle = \frac{1}{2}$, $\text{cor}\langle v_2, v_2\rangle = 1$, *and* $\text{cor}\langle v_3, v_4\rangle = 0$.

2.3 Summary Score

Based on the representative correlation, we define the representative score below.

Definition 4 (Representative Score). *The representative score of vertex v to vertex u, denoted by* $\text{rep}\langle v, u\rangle$, *is proportional to the correlation* $\text{cor}\langle v, u\rangle$ *and node weight* $\text{freq}(u)$, *i.e.,* $\text{rep}\langle v, u\rangle = \text{freq}(u) \cdot \text{cor}\langle v, u\rangle$.

The representative impact of v to u is proportional to $\text{cor}\langle v, u\rangle$. Obviously, a different representative vertex v could have significantly different representative score $\text{rep}\langle v, u\rangle$. Motivated by [14], for each vertex $u \in V$, the best representative vertex of u is the one vertex v achieving the largest representative score $\text{rep}\langle v, u\rangle$. Recall that this paper considers to select a small set of vertices S to summarize the whole DAG G. To this end, we define the summary score as follows.

Definition 5 (Summary Score). *Given a vertex set $S \subseteq V$, the summary score of S is the sum of the maximum representative scores for all vertices in G, denoted by*

$$\text{g}(S) = \sum_{u \in V} \max_{v \in S} \text{rep}\langle v, u\rangle.$$

Example 3. *In Figure 2,* $\text{rep}\langle v_2, v_5\rangle = \text{freq}(v_5) \cdot \text{cor}\langle v_2, v_5\rangle = 18 * \frac{1}{2} = 9$. *Note that* $\text{rep}\langle v_2, v_2\rangle = 0$ *as* $\text{freq}(v_2) = 0$, *and* $\text{rep}\langle v_3, v_4\rangle = 0$ *as* $\text{cor}\langle v_3, v_4\rangle = 0$. *We have the summary score* $\text{g}(\{v_2, v_4\}) = 67$ *and* $\text{g}(\{v_3, v_4\}) = 81$.

Properties of summary score $g(S)$. In the following, we first identify the essential properties that should be obeyed by a good summary score function $\text{g}(S)$, and then analyze how our summary score function in Def. 5 satisfies the required properties.

Recall that the objective of our DAG summarization is to select a small set of important vertices S to show a good summarization of a DAG G. The summary score function $\text{g}(S)$ should satisfy the following four properties:

1. Diversity. The vertices of S are diverse but not very similar;
2. Small-scale. $|S|$ is small, even within human cognitive capacity;
3. Large Coverage. The vertices of S can reach many vertices;
4. High Correlation. The representative correlation of S to a large weighted vertex is high.

Our designed function $\text{g}(S)$ in Def. 5 satisfies above four properties. First, one vertex could be represented by only one element in S by Def. 5. Thus, if the representative vertices in S are similar, the summary score may be low. The maximum objective function of $\text{g}(S)$ leads to a diverse set of S. Second, a small number k can be easily set up to control the limit of summarized vertices, which could guarantee the small-scale property. Last but not least, according to the properties of summary score in Def. 5 and representative score in Def. 4, $\text{g}(S)$ satisfies the large coverage and high correlation criteria respectively. For example, in Figure 1(c), our summary result by $\text{g}(S)$ covers all diseases and has high correlation of these important diseases with large weights in Figure 1(a).

2.4 Problem Formulation

The problem of graph summarization in a directed acyclic graph (kDAG-problem) studied in this paper is formulated as follows.

kDAG-problem. Given a DAG $G = (V, E, \text{freq})$ and an integer k, the problem is finding a set of representative vertices $S \subseteq V$, to achieve the maximum summary score $\text{g}(S)$ with $|S| = k$, i.e.

$$S^* = \arg \max_{S \subseteq V, |S|=k} \text{g}(S).$$

Example 4. *Consider a* DAG *G shown in Figure 2 and an integer $k = 2$, the optimal solution of* kDAG-problem *is $S^* = \{v_3, v_4\}$ where $g(S^*) = 81$ is larger than all the other summary scores.*

3 HARDNESS AND PROPERTY ANALYSIS

In this section, we analyze the hardness of our problem and the property of summary score function.

3.1 Hardness

Theorem 1. *The* kDAG-problem *is NP-hard.*

Proof. We reduce the well-known NP-complete 3-SAT problem to our problem. Given a set of literals $\mathcal{X} = \{x_1, \bar{x}_1, \cdots, x_m, \bar{x}_m\}$ where $m \geq 1$ and some boolean clauses $C = \{c_1, ..., c_l\}$ where $l \geq 1$, and each clause contains 3 literals from $\mathcal{X}$ in conjunctive normal form, i.e., $c_p = x_i \cup x_j \cup x_k$. It is known that checking whether there exists a satisfying boolean assignment of variables $x_1, ..., x_m$ to make the CNF $\mathcal{F} = c_1 \cap ... \cap c_l$ as true is NP-complete [17].

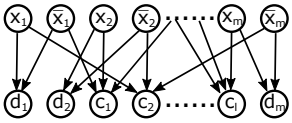


Figure 3: An illustration of the construction of a kDAG-problem **instance from a 3-SAT instance.**

Given an instance of 3-SAT with formula $\mathcal{F}$ and literals $\mathcal{X}$, we construct an instance of the problem of checking whether exists an optimal solution of kDAG-problem where $k = m$ achieving the score of $110ml + l$ in graph G as follows. We first construct $2m$ isolated nodes of $\mathcal{X}$ and other l isolated nodes C in G. For each clause $c_p = x_i \cup x_j \cup x_k$, we insert three directed edges $\langle x_i, c_p\rangle$, $\langle x_j, c_p\rangle$, and $\langle x_k, c_p\rangle$ into G. In addition, we add m nodes $\mathcal{D} = \{d_1, ..., d_m\}$ and respectively two edges $\langle x_i, d_i\rangle$, and $\langle \bar{x}_i, d_i\rangle$ for each node $d_i \in \mathcal{D}$. Now, we have a completed DAG $G = (V, E, \text{feq})$ where $V = \mathcal{X} \cup C \cup \mathcal{D}$ and the weights of vertex v is designed as follows:

$$\text{feq}(v) = \begin{cases} 100l, & if\ v \in \mathcal{X} \\ 20l, & if\ v \in \mathcal{D} \\ 2, & if\ v \in C. \end{cases}$$

Figure 3 shows an example of the constructed G and frequency function feq. Let the kDAG-problem be to find S achieving the maximum score $\text{g}(S) = \sum_{y \in V} \max_{x \in S \cap \text{anc}_G(y)} \{\text{feq}(y) \cdot \text{cor}\langle x, y\rangle\}$ with $|S| = m$. The hardness follows from this.

($\Leftarrow$) : Suppose $S^* \subseteq \mathcal{V}$ is a YES-instance of kDAG-problem, i.e., $\text{g}(S^*) = 110ml + l$ and $|S^*| = m$. We have three observations. First, one lower bound of the optimal solution S^* is $m \cdot 100l$, indicating that all m nodes should be selected from $\mathcal{X}$, i.e., $S^* \subseteq \mathcal{X}$; otherwise, if $S^* \cap (C \cup \mathcal{D}) \neq \emptyset$, the score $\text{g}(S^*) < m \cdot 100l$. Second, another tight lower bound of the optimal solution is $m \cdot (100l + 10l) = 110ml$, indicating that only one node in each pair of nodes x_i and $\bar{x}_i$ is selected in answer S^*; Otherwise, if there would exist $x_i \in S^*$ and $\bar{x}_i \in S^*$, $\text{g}(S^*) < m \cdot 110l$. Finally, the optimal solution achieves $\text{g}(S^*) = 110ml + l$, indicating that each node $c \in C$ is covered by at least one variable $x \in S^* \subseteq \mathcal{X}$. Let each variable $x \in S^*$ be true, and then each clause $c \in C$ is shown to be true. As a result, $\mathcal{F} = c_1 \cap ... \cap c_l = true$, and there exists a boolean value assignment of $\mathcal{X}$ following by S as a YES-instance of the 3-SAT problem.

($\Rightarrow$) : Suppose a boolean value assignment $h : \mathcal{X} \to \{true, false\}$ is a YES-instance of the 3-SAT problem, i.e., $\mathcal{F} = c_1 \cap ... \cap c_l = true$. Since $\mathcal{F} = true$, we can obtain $c_p = true$ for $1 \leq p \leq l$. Thus, for each $c_p = x_i \cup x_j \cup x_k$, there exists at least one variable $x' = true$, where $x' \in \{x_i, x_j, x_k\}$, and the edge $\langle x', c_p\rangle$ exists in G. Due to the mutual exclusion of function h, if $h(x_i) = true$, then $h(\bar{x}_i) = false$; otherwise, if $h(x_i) = false$, thus $h(\bar{x}_i) = true$. Based the above facts, we can select exactly m variables of $\mathcal{X}$ satisfying the true assignment and use S to represent it. In the graph G, the summary score for S is $100ml$. In addition, for each pair of x_i and $\bar{x}_i$, there exists exactly one variable in S, and node d_i can be covered in G. Thus, the summary score for $\mathcal{D}$ is $m \cdot 20l \cdot \frac{1}{2} = 10ml$. Also, the summary score for C is $l \cdot 2 \cdot \frac{1}{2} = l$. Overall, the total score is $110ml + l$. This shows that there exists a summary set S that is a YES-instance of kDAG-problem. □

3.2 Property of g(S) and Correlation Functions

Monotonicity and Submodularity of g(S). We show that our summary score $\text{g}(S)$ is monotone and submodular. A set function $f : 2^U \to \mathbb{R}^{\geq 0}$ is said to be monotone provided for sets S_1 and S_2 with $S_1 \subseteq S_2 \subseteq V$, $f(.)$ is monotone with $f(S_1) \leq f(S_2)$. It is said to be submodular provided for all sets $S \subseteq T \subseteq U$ and element $x \in U \setminus T$, $f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$, i.e., the marginal gain of an element has the so-called "diminishing returns" property.

Lemma 1. g(.) *is monotone and submodular.*

Proof. The proof can be similarly done as in [14, 27]. □

Alternative Functions of Correlation. Based on the analysis of Def. 3, we identify the properties of correlation.

Definition 6 (Correlation Functions). *Given two vertices x, y, the correlation function* $\text{F}(\text{dist}\langle x, y\rangle)$ *satisfies the two properties: (a) Non-negative with* $\text{F}(\text{dist}\langle x, y\rangle) \in [0, 1]$*; (b) Strictly monotonically decreasing.*

To model the representative impact of x to y, the correlation $\text{F}(\text{dist}\langle x, y\rangle)$ should be strict monotonically decreasing with the distance $\text{dist}\langle x, y\rangle$. The further is the distance from x to y, the weaker is the correlation. Following the properties, we design alternative correlation functions as follows.

1. $\text{F}_1(\text{dist}\langle x, y\rangle) = 1 - \frac{\text{dist}\langle x, y\rangle}{n}$, if $x \to y$; otherwise, 0.
2. $\text{F}_2(\text{dist}\langle x, y\rangle) = \frac{1}{\text{dist}\langle x, y\rangle + 1}$
3. $\text{F}_3(\text{dist}\langle x, y\rangle) = \frac{1}{e^{\text{dist}\langle x, y\rangle}}$

4 APPROXIMATE ALGORITHM

In this section, we introduce a fast approximate algorithm using greedy strategy to tackle the kDAG-problem.

4.1 Greedy+

Marginal Gain. We begin with a definition of marginal gain. Given a set of summary vertices $S \subseteq V$, a vertex $v \in V$, we examine the change of the summary score $\text{g}(S)$ by adding v into S. Note that the additional summary score is the difference between $\text{g}(S \cup \{v\})$ and $\text{g}(S)$, i.e., $\text{g}(S \cup \{v\}) - \text{g}(S)$. This motivates the following.

Definition 7 (Marginal Gain). *Given a* DAG G *and a set of summary vertices S, the marginal gain of v, denoted by $\triangle_{\text{g}}(v|S)$, is the additional summary score of $\triangle_{\text{g}}(v|S) = \text{g}(S \cup \{v\}) - \text{g}(S)$.*

Obviously, $\triangle_{\text{g}}(v|S) \geq 0$ always holds, according to the increasing monotonicity property of the $g(\cdot)$ function. For a given S, we add the vertex with the largest marginal gain into S, in order to maximally increase the total summary score. Thus, the greedy strategy of our algorithm is presented as follows. We start from an empty set $S = \emptyset$, and iteratively add one vertex v with the largest marginal gain $\triangle_{\text{g}}(v|S)$ into S, until the answer size $|S|$ reaches k.

Summary Distance. Given a vertex set $S \subseteq V$, we define the summary distance from $\mathcal{S}$ to a vertex $y \in V$, denoted by $\text{dist}(S, y)$, as the minimum distance from the vertex $v \in \mathcal{S}$ to y:

$$\text{dist}(S, y) = \min_{v \in S} \{\text{dist}\langle x, y\rangle\} = \min_{v \in S \cap \text{anc}(y)} \{\text{dist}\langle v, y\rangle\} \quad (1)$$

Obviously, if $S \cap \text{anc}(y) = \emptyset$, the summary distance $\text{dist}(S, y) = +\infty$ holds. We define the representative score of S on a vertex $y \in V$, denoted by $\text{rep}(S, y) = \max_{v \in S \cap \text{anc}(y)} \text{rep}\langle v, y\rangle$. We have the following observation and lemma.

$$\text{rep}(S, y) = \max_{v \in S \cap \text{anc}(y)} \text{rep}\langle v, y\rangle = \frac{\text{feq}(y)}{1 + \text{dist}(S, y)} \quad (2)$$

	Node	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Step 1	$\triangle_g(x\|S)$	**37**	**46**	**39**	**42**	**18**	**6**	**6**
Step 2	$\triangle_g(x\|S)$	**1**	/	**14**	**21**	18	6	6

Table 1: The steps of Greedy+. Red indicates the largest $\triangle_g(x|S)$; Blue indicates that $\triangle_g(x|S)$ is not updated.

Lemma 2. *If* $\text{dist}(S\cup\{v\}, y) = \text{dist}(S, y)$, $\text{rep}(S\cup\{v\}, y) = \text{rep}(S, y)$.

Proof. By Eq. 2, the proof is straightforward. □

Recall that $\triangle_g(x|S)$ can be calculated as $\triangle_g(x|S) = \sum_{y\in V} \text{rep}(S\cup\{x\}, y) - \text{rep}(S, y)$. According to Lemma 2, only the vertices $y \in \text{des}(x)$ satisfying $\text{dist}(S\cup\{x\}, y) < \text{dist}(S, y)$ need to be considered in the computation of $\triangle_g(x|S)$. Noted that $\text{dist}(S\cup\{x\}, y) = \min(\text{dist}(S, y), \text{dist}\langle x, y\rangle)$. If $\text{dist}(S\cup\{x\}, y) < \text{dist}(S, y)$, $\text{dist}\langle x, y\rangle < \text{dist}(S, y)$. Thus, we can prune the vertices $y \in V$ with $\text{dist}\langle x, y\rangle = \text{dist}(S, y)$, or $y \notin \text{des}(x)$. We have the following theorem.

Theorem 2. $\triangle_g(x|S) = \sum_{y\in C_x} \text{rep}(S\cup\{x\}, y) - \text{rep}(S, y)$ *holds, where* $C_x = \{y \in \text{des}(x) : \text{dist}\langle x, y\rangle < \text{dist}(S, y)\}$.

Greedy+ Algorithm. Based on the greedy strategy and Theorem 2, we propose a fast greedy algorithm called Greedy+. The general idea is to iteratively select the vertices with the largest marginal gains and add them into answers. However, it does not recompute $\triangle_g(x|S)$ for every vertex $v \in V$ at each iteration. Only the good candidate vertices are updated with the newest $\triangle_g(x|S)$, which saves lots of computations. Moreover, we develop a faster procedure of computing $\triangle_g(x|S)$ by one traversal of C_x, instead of the whole vertex set V.

Algorithm 1 gives the details of Greedy+. The algorithm first constructs a max-heap $\mathcal{H}$ to maintain $\triangle_g(v|S)$ for all vertices v (lines 2-5). It initializes the summary distance of all vertices be $+\infty$ due to $S = \emptyset$. The algorithm also computes $\triangle_g(v|S)$ for every vertex v and pushes them into $\mathcal{H}$ (lines 3-5). Then, Algorithm 1 starts the selection of summary vertices (lines 6-12). It gets $(x^*, \triangle_{max})$ from the max-heap $\mathcal{H}$ where x^* is the vertex and $\triangle_{max}$ is the upper bound of the marginal gain of including all vertices (line 7). If $\triangle_{max} > \triangle_g(x^*|S)$, we need to compute $\triangle_g(x^*|S)$ and update the corresponding $(x^*, \triangle_g(x^*|S))$ in $\mathcal{H}$ (lines 8-10). This may happen due to the lazy update of $\mathcal{H}$. Otherwise, x^* is the vertex with the largest marginal gain among all candidates. The algorithm adds x^* into S and updates the summary distance $\text{dist}(v)$ for $v \in \text{des}(x^*)$.

Compute $\triangle_g(x|S)$. The procedure of computing $\triangle_g(x|S)$ is shown in Algorithm 2. The algorithm uses a breadth-first-search (BFS) to traverse the reachable vertices u from x, and sums over the difference of $\frac{\text{feq}(u)}{1+\text{dist}\langle v,u\rangle} - \frac{\text{feq}(u)}{1+\text{dist}(S,u)}$ (lines 1-10). Note that Algorithm 2 prunes the vertices u whose summary distance $\text{dist}(S, u)$ is not larger than $\text{dist}\langle x, u\rangle$ (lines 6-8), thanks to Theorem 2. Finally, the algorithm returns $\triangle_g(x|S)$ (line 11).

Example 5. *We apply Algorithm 1* Greedy+ *on a* DAG *G in Figure 2 for $k = 2$. Firstly, Algorithm 1 computes $\triangle_g(x|S)$ for each vertex x, as shown in Table 1. At the first iteration, the vertex v_2 has the maximum value of $\triangle_g(v_2|S) = 46$, so it is added into the set S. At the second iteration, the $\triangle_g(x|S)$ values of vertices v_4, v_3, v_1 are updated. It selects v_4 with the maximum $\triangle_g(v_4|S) = 21$. The vertices v_5, v_6, v_7 are not updated because their $\triangle_g(x|S)$ values are smaller than 21. Finally, the answer $S = \{v_2, v_4\}$ and the summary score $g(S) = 67$. However, this answer is not optimal. The exact solution for G is the set $S^* = \{v_3, v_4\}$ and $g(S^*) = 81$.*

Algorithm 1 Greedy+

Input: A DAG $G = (\mathcal{V}, E, \text{feq})$, an integer k.
Output: A set of k summary elements S.
1: Let $S \leftarrow \emptyset$;
2: Let $\mathcal{H}$ be a max-heap H to maintain $\triangle_g(v|S)$ for all $v \in V$;
3: **for** vertex $v \in V$ **do**
4: $\quad\text{dist}(S, v) \leftarrow \infty$;
5: $\quad\mathcal{H}$.push($(v, \triangle_g(v|S))$) by invoking Algorithm 2;
6: **while** $|S| < k$ **do**
7: $\quad(x^*, \triangle_{max}) \leftarrow \mathcal{H}$.pop();
8: $\quad$**if** $\triangle_{max} > \triangle_g(x^*|S)$ **then**
9: $\quad\quad\mathcal{H}$.push($(x^*, \triangle_g(x^*|S))$);
10: $\quad\quad$**Continue**;
11: $\quad S \leftarrow S \cup \{x^*\}$;
12: $\quad$Update $\text{dist}(S, v)$ for vertex $v \in \text{des}(x^*)$ in one DFS;
13: **return** S;

Algorithm 2 Compute $\triangle_g(x|S)$

Input: A DAG $G = (V, E, feq)$, a set of vertices S, vertex x.
Output: $\triangle_g(x|S)$.
1: Queue $Q \leftarrow \{x\}$; $Visited \leftarrow \{x\}$;
2: Distance $\text{dist}\langle x, x\rangle \leftarrow 0$;
3: Let $\triangle_g(x|S) \leftarrow 0$; //Initialization
4: **while** $Q \neq \emptyset$ **do**
5: $\quad v \leftarrow Q.pop()$;
6: $\quad$**for** vertex $u \in \text{N}^-(v)$ **do**
7: $\quad\quad\text{dist}\langle x, u\rangle \leftarrow \text{dist}\langle x, v\rangle + 1$;
8: $\quad\quad$**if** $u \notin Visited$ and $\text{dist}(S, u) > \text{dist}\langle x, u\rangle$ **then**
9: $\quad\quad\quad\triangle_g(x|S) \leftarrow \triangle_g(x|S) + (\frac{\text{feq}(u)}{1+\text{dist}\langle v,u\rangle} - \frac{\text{feq}(u)}{1+\text{dist}(S,u)})$;
10: $\quad\quad\quad Q \leftarrow Q \cup \{u\}$; $Visited \leftarrow Visited \cup \{u\}$;
11: **return** $\triangle_g(x|S)$;

4.2 Approximation and Complexity Analysis

We analyze the approximation and complexity of Algorithm 1.

Theorem 3. *Algorithm 1 achieves a* $(1 - \frac{1}{e})$*-approximation of optimal answers.*

Proof. Assume that S^* is an optimal answer and S is the result by Algorithm 1. Obviously, $g(S) \geq (1 - \frac{1}{e}) \cdot g(S^*)$ holds for maximizing a monotone submodular set function with the cardinality constraint of $|S| = k$ by Lemma 1. □

Theorem 4. *Algorithm 1 takes $O(n \cdot k \cdot m)$ time and $O(m)$ space.*

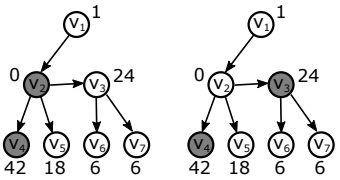
Proof. Algorithm 1 computes $\triangle_g(x|S)$ for each vertex x at most k times. Computing $\triangle_g(x|S)$ takes $O(n + m) \in O(m)$ time in worst. Thus, the update of $\triangle_g(x|S)$ for all vertices x takes $O(knm)$ time in total. Moreover, the heap $\mathcal{H}$ takes $O(nk \log n)$ time for the update of push operations and pop operations. And it takes $O(k \cdot m)$ time for updating the distance in a total of k rounds. As a result, Algorithm 1 takes $O(km+nk \log n+nkm) \subseteq O(n\cdot k\cdot m)$ time and $O(m)$ space. □

5 AN EFFECTIVE SOLUTION ON DAGS

In this section, we develop an effective algorithm EXT-Greedy for kDAG-problem based on Greedy+ and subtree extraction, which achieves high-quality answers no worse than Greedy+.

Algorithm 3 EXT-Greedy

Input: A DAG $G = (V, E, \text{feq})$, an integer k.
Output: A set of k summary elements S.
1: $S^+ \leftarrow$ Apply the Greedy+ in Algorithm 1 on G;
2: $T_{S^+} \leftarrow$ Extracts subtree T_{S^+} of G using S^+;
3: $S \leftarrow$ Apply the DP approach [32] on T_{S^+} for top-k summarization;
4: **return** S;



(a) Subtree T_{S^+} (b) DP answer

Figure 4: An example of EXT-Greedy **algorithm.**

5.1 EXT-Greedy

W.l.o.g, assume that a DAG $G(V, E, \text{feq})$ has a unique root r with $|\mathsf{N}^+(r)| = 0$. Otherwise, we add G with a virtual root r and a few directed edges $\langle r, u\rangle$ for all vertices u with $|\mathsf{N}^+(u)| = 0$. We define a subtree $T(V, E_t, \text{feq})$ of G as a directed tree rooted by r with edges $E_t \subseteq E$. Throughout the remaining paper, we use $\mathrm{g}_G(S)$ to represent the summary score of S in DAG G.

Overview. The idea of EXT-Greedy is to construct a tree T based on an approximate answer of k representative vertices produced by Algorithm 1. A directed tree is a special instance of DAG. There exists a dynamic programming (DP) approach in polynomial time to optimally tackle DAG summarization in a directed tree [32]. Therefore, EXT-Greedy applies a tree-based dynamic programming method on the extracted tree T to obtain an improved answer.

Subtree extraction. Given a DAG G rooted by r and a summary set S produced by Algorithm 1, we can extract a directed tree T rooted by r from G. The generated tree T keeps the same distance oracle for summary vertices S in G. In other words, the summary distance $\text{dist}(S, v)$ is unchanged in T for all vertices $v \in V$.

EXT-Greedy algorithm. The method of EXT-Greedy is detailed in Algorithm 3. It first applies the the Greedy+ (Algorithm 1) on G, which finds a set of k summary vertices S^+ (line 1). Then, it extracts a subtree T_{S^+} of G using S^+ . Finally, Algorithm 3 invokes an exact algorithm of DP [32] to find a summary set S in the tree T_{S^+}, and returns S as the answer (lines 3-4).

Example 6. *Figure 4 shows an example of applying* EXT-Greedy *on G in Figure 2. The answer of* Greedy+ *as $S^+ = \{v_2, v_4\}$ and $\mathrm{g}(S^+) = 67$. Based on S^+,* EXT-Greedy *extracts a subtree T_{S^+} in Figure 4(a). Figure 4(b) shows the optimal solution $S = \{v_3, v_4\}$ in T_{S^+} by* DP. *In the extracted tree T_{S^+}, the summary score $\mathrm{g}_{T_{S^+}}(S) = 72 \geq 67 = \mathrm{g}(S^+)$. S is an optimal solution of G with the largest summary score $\mathrm{g}_G(S) = 81$ in G in Figure 2.*

5.2 Theoretical Analysis

We first analyze the quality of EXT-Greedy in Algorithm 3. Assume that S^* is the optimal answer of G and $|S^*| = k$.

Theorem 5. *Given a* DAG *G, $\exists$ a tree $T \subseteq G$ such that $\mathrm{g}_G(S^*) = \mathrm{g}_T(S)$ where S is the optimal answer for T.*

Proof. We construct a directed subtree $T(V, E, \text{feq}) \subseteq G$ as follows. We apply Algorithm 3 to extract T from G using the input of optimal answer S^*. As, $S^* \subseteq V$ and $|S^*| = k$, S^* is one feasible answer in T. Moreover, the distance oracle of S^* is the same in both T and G. DP finds the optimal answer S in a tree T, indicating $\mathrm{g}_T(S) \geq \mathrm{g}_T(S^*) = \mathrm{g}_G(S^*)$. Obviously, $\mathrm{g}_T(S) \leq \mathrm{g}_G(S) \leq \mathrm{g}_G(S^*)$, as $T \subseteq G$. As a result, $\mathrm{g}_G(S^*) = \mathrm{g}_T(S)$ holds. □

Theorem 5 shows that our algorithm EXT-Greedy probably find an optimal solution of G. In the following, we prove that EXT-Greedy finds an answer no worse than Greedy+.

Theorem 6. *$g_G(S^+) = g_{T_{S^+}}(S^+) \leq g_{T_{S^+}}(S) \leq g_G(S)$ holds, where S^+ and S are respectively the answers of Algorithm 1 and Algorithm 3.*

Proof. First, we prove $g_{T_{S^+}}(S^+) = g_G(S^+)$. As $T \subseteq G$, vertices have the same weights in T and G. Moreover, T and G has the same distance oracle for summary vertices S^+. $g_{T_{S^+}}(S^+) = g_G(S^+)$ holds. Second, we prove $g_{T_{S^+}}(S^+) \leq g_{T_{S^+}}(S)$. In tree T_{S^+}, DP algorithm finds an optimal solution S with the largest $g_{T_{S^+}}(S)$, i.e., $g_{T_{S^+}}(S) \geq g_{T_{S^+}}(S^+)$. Third, we prove $g_{T_{S^+}}(S) \leq g_G(S)$. Since $T \subseteq G$, $g_{T_{S^+}}(S) \leq g_G(S)$ clearly holds. □

6 K-PCGS: PRUNING AND COMPRESSION

In this section, we propose a fast DAG summarization algorithm k-PCGS, which prunes candidate vertices and compresses DAG. We first give an algorithm overview and then presents the details of carefully designed pruning and compression techniques. Finally, we propose variant approximate methods and analyze the complexity.

Overview. To improve Greedy+ in Algorithm 1, we develop two efficient techniques : *pruning candidates* and *compressing DAG*. The general idea of pruning candidates is to prune unnecessary vertices, in order to reduce the number of vertices whose compute $\triangle_{\mathrm{g}}(v|S)$. The general idea of compressing DAG is graph sparsification, which shrinks the original graph by discarding tree components and useless vertices.

6.1 Candidate Pruning

Candidate Set. Let C be a candidate set of vertices for top-k DAG summarization and $C \subseteq V$. Greedy+ in Algorithm 1 considers all vertices V as candidates (lines 3-5) and $C = V$. However, some vertices have small summary score contributions, which are impossible qualified for the top-k answer and can be early pruned, e.g. vertices v_5, v_6 and v_7 in Figure 2 for $k = 2$ in the above example.

Upper and Lower Bounds of $\triangle_{\mathrm{g}}(v|S)$. Let $C = V$ and see how to reduce C using the upper and low bounds of summary score contribution as follows.

For each vertex v, we define an upper bound $\mathsf{UB}(v)$ and a lower bound $\mathsf{LB}(v)$ of its summary score contribution $\triangle_{\mathrm{g}}(v|S)$, such that $\mathsf{LB}(v) \leq \triangle_{\mathrm{g}}(v|S) \leq \mathsf{UB}(v)$ for any set $S \subseteq C \setminus \{v\}$ and $|S| < k$.

Lemma 3. *For a vertex $v \in C$, $\mathsf{UB}(v) = \triangle_{\mathrm{g}}(v|\emptyset)$, then $\mathsf{UB}(v) \geq \triangle_{\mathrm{g}}(v|S)$ holds for any set $S \subseteq C \setminus \{v\}$ and $|S| < k$.*

Proof. Based on Lemma 1 and $\emptyset \subseteq S$, $\triangle_{\mathrm{g}}(v|\emptyset) = \mathrm{g}(\{v\}) - \mathrm{g}(\emptyset) \geq \mathrm{g}(\{v\} \cup S) - \mathrm{g}(S) = \triangle_{\mathrm{g}}(v|S)$ for any set $S \subseteq C$ and $|S| < k$. □

Lemma 4. *For a vertex $v \in C$, $\mathsf{LB}(v) = \triangle_{\mathrm{g}}(v|C \setminus \{v\})$, then $\mathsf{LB}(v) \leq \triangle_{\mathrm{g}}(v|S)$ holds for any set $S \subseteq C \setminus \{v\}$ and $|S| < k$.*

Proof. Based on Lemma 1 and $S \subseteq C \setminus \{v\}$, $\triangle_g(v|C \setminus \{v\}) = g(C) - g(C \setminus \{v\}) \leq g(\{v\} \cup S) - g(S) = \triangle_g(v|S)$ for any set $S \subseteq C$ and $|S| < k$. □

Candidate Pruning. Beside the upper and lower bounds of vertices, we define a minimum threshold for vertices to be candidate answers. Specifically, we define a pruning threshold $\Delta_k^* = \{\mathsf{LB}(v^*) : v^* \in C$ is a vertex with the k-th largest lower bounds in $C\}$. In other words, there exist no less than k vertices $u \in C$ with $\mathsf{LB}(u) \geq \Delta_k^*$ and less than $|C| - k$ vertices $u \in C$ with $\mathsf{LB}(u) < \Delta_k^*$.

Theorem 7 (Pruning Candidates). *Given a vertex $v \in C$, if $\mathsf{UB}(v) < \Delta_k^*$, the vertex v can be removed from C.*

Proof. If v is an answer of Algorithm 1, there exists a set $S \subseteq C$ that satisfies $|S| = k - 1$ and $\triangle_g(v|S) \geq \triangle_g(u|S)$ for each $u \in C \setminus S$. So $\mathsf{UB}(v) \geq \triangle_g(v|S) \geq \triangle_g(u|S) \geq \mathsf{LB}(u)$ for each vertex $u \in C \setminus S$. There exists *at most k - 1* vertices $u \in C$ satisfying $\mathsf{UB}(v) < \mathsf{LB}(u)$, and $u \in S$ must hold. However, $\mathsf{UB}(v) < \Delta_k^*$ indicates that there exist *at least k* vertices $u \in C$ satisfy $\mathsf{UB}(v) < \mathsf{LB}(u)$, which is a contradiction. □

Based on Theorem 7, Lemmas 3 and 4, we can delete all the candidates $v \in C$ for $\mathsf{UB}(v) < \Delta_k^*$. Briefly, we can delete the vertices that are impossible to be answer in Greedy+. In addition, the lower bounds and candidate C are dynamically updated and affected by each other. Specifically, when vertices v having $\mathsf{UB}(v) \leq \Delta_k^*$ are removed from C_1, C_1 shrinks into a smaller set $C_2 \subset C_1$. Thus, the lower bounds $\mathsf{LB}(v)$ increases as $\triangle_g(v|C_2 \setminus \{v\}) \geq \triangle_g(v|C_1 \setminus \{v\})$ for $C_2 \subset C_1$. Accordingly, Δ_k^* increases largely, which can further prune more vertices and lead smaller candidates. The iterative process stops when $\mathsf{UB}(u) \geq \Delta_k^*$ for any $u \in C$. Note that Theorem 7 is applicable to any kinds of lower bounds and upper bounds of $\triangle_g(v|S)$.

6.2 DAG Compression

We define a compressed graph $G_c(V_c, E_c, \mathsf{Feq}_c)$ of $G(V, E, \mathrm{feq})$ where $V_c \subseteq V$, $E_c \subseteq E$, and Feq_c is an aggregate frequency function of vertices. An aggregate frequency set of vertex v is $\mathsf{Feq}_c(v) = \{(d, \mathrm{feq}_d(v)) : d \in \mathbb{Z}^{\geq 0}, \mathrm{feq}_d(v) \in \mathbb{R}^{\geq 0}\}$, where the aggregate frequency $\mathrm{feq}_d(v) = \sum_{\mathsf{dist}\langle v,u\rangle = d} \mathrm{feq}(u)$ for $d \in [0, h(v)]$ and $h(v) = \max_{u \in \mathsf{des}(v)} dist\langle v, u\rangle$. Before DAG compression, all aggregate frequency sets are initialized as $\mathsf{Feq}_c(v) = \{(0, \mathrm{feq}(v))\}$.

We compress the DAG into a new graph G_c by graph removal. When one vertex is deleted, the vertex and its incident edges are deleted from graph together. There exists two kinds of disqualified nodes v that could be removed: one has $\mathsf{UB}(v) < \Delta_k^*$ in the tree-shape component and the other has $\mathsf{UB}(v) = 0$. Given a vertex v, we say v is located in the tree-shape component of G if and only if the induced subgraph of G by the vertex set $\mathsf{des}(v)$ is a data structure of tree rooted by v. After a vertex deletion of v, to ensure no loss of vertex importances, we transform the weight importances of v to its in-neighbor $u \in \mathsf{N}^+(v)$ and update the corresponding aggregate frequency $\mathsf{Feq}_c(u)$ as follows.

$$\mathsf{Feq}_c(u) = \{(d, \mathrm{feq}_d(u) + \mathrm{feq}_{d-1}(v)) : d \in [1, h(u)]\} \cup \{(0, \mathrm{feq}(u))\} \tag{3}$$

In this way, we store the frequency of its deleted out-neighbor v in $\mathsf{Feq}_c(u)$ for $v \in \mathsf{N}^-(u)$.

Algorithm 4 k-PCGS

Input: A DAG $G = (V, E, feq)$, an integer k.
Output: A summary set of k vertices S.

```
1: Let T be a k-size set to maintain the pruning threshold Δ*_k.
2: Let H be a min-heap to maintain ub(v) for vertices v with |N⁻(v)| = 0;
3: Initialization: candidate set C ← V; T ← ∅;
4: for vertex v ∈ C do
5:     lbFeq(v) ← {(0, feq(v))}; lb(v) ← feq(v)/2 in Eq.5;
6:     ubFeq(v) ← {(0, feq(v))}; ub(v) ← feq(v) in Eq.4;
7:     Let be an aggregate frequency set Feq_c(v) ← {(0, feq(v))};
8:     if |T| < k then
9:         T ← T ∪ {v};
10:    else if lb(v) > min_{v∈T}{lb(v)} then
11:        u* ← arg min_{v∈T}{lb(v)};
12:        T ← T \ {u*} ∪ {v};
13:    |N⁻(v)| ← |{⟨v, u⟩ ∈ E}|;
14:    if |N⁻(v)| = 0 then
15:        H.push((ub(v), v));
16: while H ≠ ∅ do
17:    (ub(v), v) ← H.pop();
18:    Δ*_k ← min_{v∈T,|T|=k}{lb(v)};
19:    if ub(v) < Δ*_k then
20:        Candidate pruning: C ← C \ {v} using Theorem 7;
21:    if ub(v) = 0 then
22:        Delete vertex v and its incident edges from G;
23:    for vertex u ∈ N⁺(v) do
24:        |N⁻(u)| = |N⁻(u)| − 1;
25:        Update ubFeq(u) and ub(u) in Eq. 4;
26:        if |N⁺(v)| = 1 and ub(v) < Δ*_k then
27:            Update lbFeq(u) and lb(u) using Eq. 5;
28:            Maintain T using the new lb(u) accordingly;
29:            if des(v) = {v} then
30:                Transfer weights and update Feq_c(u) using Eq. 3;
31:                Delete vertex v and its incident edges from G;
32:        if |N⁻(u)| = 0 then
33:            H.push((ub(u), u));
34: Let the shrank graph G be a compressed graph G_c;
35: Find the top-k answer S from candidates C on G_c using Algorithm 1.
```

In addition, we directly delete from graph G the vertices v whose $\mathsf{UB}(v) = 0$. Obviously, these vertices and their descendants all have the weight $\mathrm{feq}(u) = 0$ for $u \in \mathsf{des}(v)$, which are not kept in the compressed graph G_c.

6.3 k-PCGS Algorithm

In the following, we present our algorithm k-PCGS based on the candidate **P**runing and DAG **C**ompression for top-**k** **G**raph **S**ummarization. An intuitive implementation of our algorithm is to first compute $\mathsf{UB}(v)$ and $\mathsf{LB}(v)$ for all vertices $v \in V$, and then apply the pruning threshold Δ_k^* for pruning candidate set C by Theorem 7 and compressing DAG as G_c. However, the exact computation of all $\mathsf{UB}(v)$ and $\mathsf{LB}(v)$ based on C is very time costly, due to the large graph size and iteratively changed C. To improve the efficiency, we integrate two processes of candidate pruning and DAG compression into one whole procedure and develop two new dynamic updating bounds for cost saving.

Dynamic Updating Bounds. We design two loose bounds $\mathsf{ub}(v)$ and $\mathsf{lb}(v)$ where $\mathsf{ub}(v) \geq \mathsf{UB}(v)$ and $\mathsf{lb}(v) \leq \mathsf{LB}(v)$.

For the upper bound, we create an aggregate frequency set $\mathsf{ubFeq}(v) = \{(d, \mathsf{ubFeq}_d(v))\}$ for each vertex v in compressed graph G_c, which has the same structure and initialization as $\mathsf{Feq}_c(v)$ but with a different updating strategy. Specifically, we transfer the weight importance $\mathsf{ubFeq}(u)$ to **all** in-neighbors $v \in N^+(u)$ as Eq. 3. The upper bound $\mathsf{ub}(v)$ is calculated by:

$$\mathsf{ub}(v) = \sum_{d=0}^{h(v)} \mathsf{ubFeq}_d(v) \cdot \frac{1}{d+1}. \tag{4}$$

We have $\mathsf{ub}(v) \geq \sum_{u \in \mathsf{des}(v)} \frac{\mathsf{feq}(u)}{1+\mathsf{dist}\langle v,u\rangle} = \triangle_{\mathsf{g}}(v|\emptyset) = \mathsf{UB}(v)$ in Lemma 3.

For the lower bound, we also create an aggregate frequency set $\mathsf{lbFeq}(v) = \{(d, \mathsf{lbFeq}_d(v))\}$. Differently, we just update the **removed** vertex v to its in-neighbor vertex. Moreover, in the updating process, we only transfer the weight importance $\mathsf{lbFeq}(u)$ as Eq. 3 to the **unique** parent v , i.e., $v \in N^+(u)$ and $|N^+(u)| = 1$. The lower bound $\mathsf{lb}(v)$ is calculated by:

$$\mathsf{lb}(v) = \sum_{d=0}^{h(v)} \mathsf{lbFeq}_d(v) \cdot (\frac{1}{d+1} - \frac{1}{d+2}). \tag{5}$$

We have $\mathsf{lb}(v) \leq \sum_{u \in \mathsf{des}(v)} \mathsf{rep}(C, u) - \mathsf{rep}(C \setminus \{v\}, u) = \triangle_{\mathsf{g}}(v|C \setminus \{v\}) \leq \mathsf{LB}(v)$ in Lemma 4.

For $\mathsf{ub}(v)$ and $\mathsf{lb}(v)$, we can obtain them for all $v \in V$ using one bottom-up scan of G, which is very efficient.

k-PCGS algorithm. Algorithm 4 presents the details of top-k summarization method based on pruning candidates and compressing DAG. It starts from the leaf nodes v with $|N^-(v)| = 0$ in a bottom-up search manner to remove disqualified vertices and update bounds $\mathsf{ub}(v)$ and $\mathsf{lb}(v)$ iteratively. Note that a vertex v with $|N^-(v)| = 0$ indicates that the bounds are updated exactly in Eq.4 and Eq.5. The algorithm first constructs a k-sized set $\mathcal{T}$ to maintain the top-k lower bound vertices in candidate set C and a priority queue $\mathcal{H}$ to maintain upper bounds $\mathsf{ub}(v)$ for all vertices v with $|\mathsf{N}^-(v)| = 0$(lines 1-3). Then, it initializes the upper bounds and low bounds of $\mathsf{ub}(v)$ and $\mathsf{lb}(v)$ for all vertices $v \in C = \mathcal{V}$ in Eq.4 and Eq.5 (lines 4-15). $\mathcal{T}$ is updated with the largest k lower bounds $\mathsf{lb}(v)$ (lines 8-12). The algorithm pushes into $\mathcal{H}$ the leaf nodes v with $|N^-(v)| = 0$ (lines 13-15). Next, it iteratively finds a vertex v with the smallest $\mathsf{ub}(v)$ in H and check whether to remove u from candidate set C and graph G (lines 16-33). It calculates the minimum threshold Δ_k^* based on $\mathcal{T}$ (line 18). If $\mathsf{ub}(v) < \Delta_k^*$, the algorithm removes v from C (lines 19-20). If $\mathsf{ub}(v) = 0$, it deletes v and its incident edges from G (lines 21-22). For each in-neighbor $u \in N^+(v)$, it then updates $\mathsf{lb}(u)$ and $\mathsf{ub}(b)$ accordingly (lines 23-31). If vertex v is a leaf node in G with $\mathsf{ub}(v) < \Delta_k^*$ and $|\mathsf{N}^+(v)| = 1$, it deletes v and its incident edges from G, and also transfers the aggregate frequency sets from v to its unique parent u. Afterwards, it checks whether to add the vertex u with $|N^-(v)| = 0$ into $\mathcal{H}$ for candidate verification (lines 32-33). Finally, we represent the reduced graph as a compressed graph G_c and apply Algorithm 1 on G_c for top-k summary results (lines 33-35).

Approximation and Complexity Analysis. In most applications, $\mathsf{ub}(v)$ is much larger than $\mathsf{UB}(v)$, and $\mathsf{lb}(v)$ is much less than $\mathsf{LB}(v)$. Thus, we set a parameter $\alpha \geq 1$ and relax Theorem 7 to delete candidates v if $\mathsf{ub}(v) \leq \alpha \cdot \Delta_k^*$ (for the lines 19 and 24 in Algorithm 4). Note that when $\alpha = 1$, k-PCGS obtains the same solution as Greedy+. We analyze the complexity of k-PCGS in Algorithm 4. Let h be the maximum of depth in G and $m_c = |E_c|$ is the size of edges in G_c. The compression and pruning steps take $O(mh)$ time to update ub(.) and lb(.), $O(n \log k)$ time to maintain $\mathcal{T}$ and $O(n \log n)$ time to maintain $\mathcal{H}$. The Algorithm 1 applied on G_c takes $O(k|C|m_c)$ time, where $m_c \leq m$. Overall, k-PCGS takes $O(nh + m)$ space in the time complexity of $O(mh + n \log n + k|C|m_c)$, which is much faster than $O(knm)$ in Algorithm 1 for $h \leq n$ and $|C| \leq n$.

Name	n	m	h	Name	n	m	h
LATT	4,226	8,190	18	IMAGE	73,298	74,732	20
LNUR	4,226	8,190	18	YAGO	5,699,254	17,512,754	17
ANIM	15,135	24,498	11	UCI	58,790,783	92,208,196	17

Table 2: The statistics of DAG datasets.

7 EXPERIMENTS

In this section, we conduct experiments to evaluate all algorithms.

Datasets. We test six real-world datasets of DAGs with hierarchical terminologies. The first two datasets, LATT and LNUR, are extracted from the Medical Entity Dictionary (MED) [16]. They have the same topological structure but with different vertex weights. The third dataset, ANIM, is extracted from the "Anime" catalog in Wikipedia[25]. The weight of each vertex is the number of page-views in one month. The next dataset, IMAGE, is extracted from the Image-net [7]. The node weight is the number of pictures in the catalog. The last two datasets, YAGO [20] is a knowledgebase from multilingual wikipedias and UCI [23] is a social friendship network extracted from Facebook. We assgin random weights to vertices following the weight distribution of LATT.

Methods compared. We compare our proposed model and algorithms with state-of-the-art methods respectively for DAGs [1]. First, we evaluate our summarization model of kDAG-problem and compare it with three state-of-the-art methods – FEQ, CAGG[16] and LASP[8]. Here, FEQ is a baseline approach, which selects k nodes with the highest frequencies[16]. CAGG is a statistical method using two metrics of aggregate frequency and contribution ratio[16]. LASP is a greedy method of finding a rooted tree, which adds the largest averaged score path until the answer size reaches k[8]. We also compare our approaches of Greedy+ (Algorithm 1), EXT-Greedy (Algorithm 3), and k-PCGS (Algorithm 4).

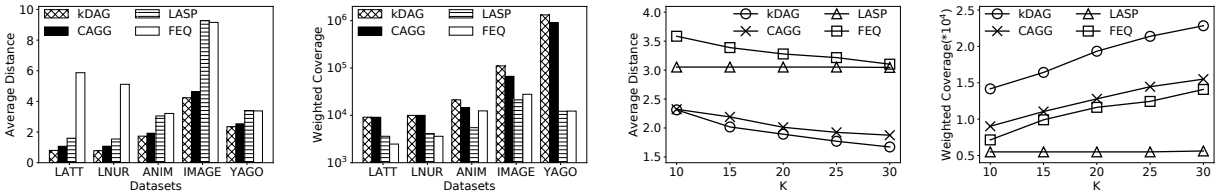
Evaluation metrics. We compare all methods in terms of effectiveness and efficiency metrics. To evaluate the quality of summary result S found by an algorithm, we use three metrics: summary score $\mathsf{g}(S)$ in Def. 5, the average distance $\mathsf{Dist}_{\mathsf{avg}}(S)$, and the weighted coverage $\mathsf{Cover}_{\mathsf{w}}(S)$. The $\mathsf{Dist}_{\mathsf{avg}}(S)$ is defined as the weighted average distance from S to all vertices in V, i.e.

$$\mathsf{Dist}_{\mathsf{avg}}(S) = \frac{\sum_{u \in V} \mathsf{dist}(S, u) \cdot \mathsf{feq}(u)}{\sum_{u \in V} \mathsf{feq}(u)}.$$

Note that if $\mathsf{dist}(S, u) = \infty$, we replace $\mathsf{dist}(S, u)$ to the length of the longest path. Moreover, the weighted coverage $\mathsf{Cover}_{\mathsf{w}}(S)$ is defined as the total weight of the vertices within 1-hop of S as $C(S) = \{u \in V : \mathsf{dist}(S, u) \leq 1\}$, denoted by

$$\mathsf{Cover}_{\mathsf{w}}(S) = \sum_{u \in C(S)} \mathsf{feq}(u).$$

[1] https://github.com/csxlzhu/CIKM20



(a) Average Distance $\text{Dist}_{\text{avg}}(S)$ (b) Weighted Coverage $\text{Cover}_{\text{w}}(S)$ (c) Average Distance $\text{Dist}_{\text{avg}}(S)$ (d) Weighted Coverage $\text{Cover}_{\text{w}}(S)$

Figure 5: Quality evaluation of all methods on DAG datasets in Figure 5(a)-(b) and ANIM dataset varied by k in Figure 5(c)-(d).

Datasets	LATT	LNUR	ANIM	IMAGE	YAGO
Greedy+	**5,747**	**6,710**	18,972	547,098	**12,602,525**
k-PCGS ($\alpha = 1$)	**5,747**	**6,710**	18,972	547,098	**12,602,525**
k-PCGS ($\alpha = 2$)	5,726	**6,710**	18,972	547,098	**12,602,525**
k-PCGS ($\alpha = 5$)	4,681	5,418	18,972	547,098	**12,602,525**
EXT-Greedy	**5,747**	**6,710**	**19,140**	**547,182**	/

Table 3: Summary scores of all algorithms. Here $k = 25$.

Datasets	LATT	LNUR	ANIM	IMAGE	YAGO	UCI
Greedy+	4,226	4,226	15,135	73,298	5,699,254	58,790,783
k-PCGS ($\alpha = 1$)	259	248	238	3,748	982	1,916,452
k-PCGS ($\alpha = 2$)	118	124	120	241	601	1,794,678
k-PCGS ($\alpha = 5$)	51	54	38	64	328	1,657,458

Table 4: The size of candidate C. Here $k = 25$.

Note that the smaller $\text{Dist}_{\text{avg}}(S)$ is, the better quality is. The larger $\text{Cover}_{\text{w}}(S)$ and $\text{g}(S)$ are, the better qualities are. To evaluate the efficiency of algorithms, we report the running time. We treat the running time as infinite if the algorithm runs exceeding 3 hours. We set the parameter $k = 25$ and $\alpha = 2$ in k-PCGS by default.

EXP-1: Quality evaluation of summarization models. We evaluate the quality of summary results by all methods on DAG datasets. Figure 5(a)-(b) show the results of average distance and weighted coverage by four competitive methods kDAG, FEQ, CAGG and LASP. Figure 5(c)-(d) depict the results of average distance and weighted coverage by varying k. Our kDAG model clearly outperforms the competitors FEQ, CAGG, and LASP, in terms of both metrics with the largest $\text{Cover}_{\text{w}}(S)$ and smallest $\text{Dist}_{\text{avg}}(S)$.

EXP-2: Quality evaluation of our algorithms. We conduct the quality evaluation of our algorithms. Table 3 shows the summary score of Greedy+ and EXT-Greedy and k-PCGS. EXT-Greedy consistently performs no worse than other methods. Note that EXT-Greedy cannot finish within the time limit on YAGO. Compared with Greedy+, k-PCGS always gets the same solution for $\alpha = 1$ and competitive results for $\alpha = 2$ and 5.

EXP-3: Approximation improvement on small DAGs. In this experiment, we evaluate the approximation of our algorithms EXT-Greedy and Greedy+. We randomly generate 300 small-scale DAGs with 20 nodes. Figure 7(a) shows the summary score of two methods on 300 DAGs. As we can see that EXT-Greedy wins Greedy+ in most cases, reaching nearly 80% of all cases. For those cases that EXT-Greedy produces no optimal results, EXT-Greedy gets at least the same summary score as Greedy+.

EXP-4: Efficiency evaluation. We evaluate the running time of different algorithms on ANIM, IMAGE, YAGO and UCI. We compare four methods of Greedy+, EXT-Greedy, k-PCGS, and CAGG. Figure 6 shows the running times of all methods by varying k. k-PCGS runs best among all methods on all datasets, and EXT-Greedy is the worst. Although CAGG achieves the competitve quality as our model, but CAGG runs slower than k-PCGS. Note that the running time of FEQ and LASP are not reported due to their poor quality.

EXP-5: Evaluation of candidate pruning. We evaluate the effectiveness of candidate pruning in k-PCGS algorithm. Table 4 shows the size of candidate set $|C|$. k-PCGS reduces the candidate size very significantly. It reduces at least 90% candidates in the worst case and more than 99% in most cases. Moreover, $|C|$ becomes smaller with the increased α.

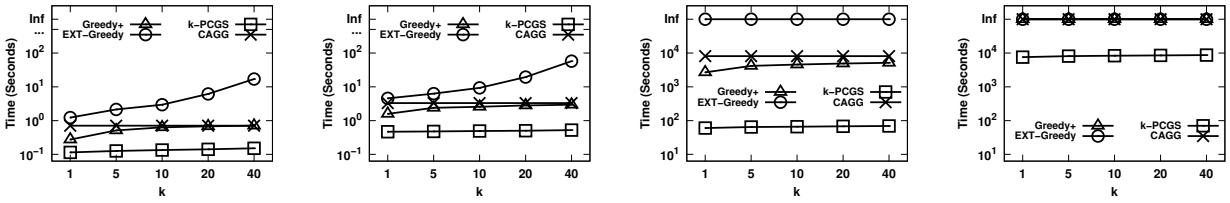
EXP-6: Scalability test. We conduct the scalability test of our algorithms varying by the size of DAGs. We randomly generate 5 DAGs with the size of nodes varying from 10^5 to 10^6. The graph statistics follow Image-net. Figure 7(b) shows that EXT-Greedy, Greedy+ and k-PCGS scale well with the increased node size. k-PCGS achieves the best peroformance with a stable scalability.

EXP-7: Usability test. We conduct a usability test of topic recommendation that verifies users' preference for four methods FEQ, CAGG, LASP and kDAG. The task is to recommend the top-k most attractive topics of SIGMOD'19. Based on the Proceedings of SIGMOD'19, we built a hierarchical DAG using 22 session topics and 104 secondary topics, where each paper belongs to a session topic and is weighted by the number of downloads recorded in ACM Digital Library [3] (up to Dec 1, 2019). We ask 15 users, who published PVLDB/SIGMOD papers in recent three years, to recommend the top-k ($k = 3, 5, 8$) most attractive topics. We evaluate an accuracy rate of overlapping topics between users' choices and methods' selections. Figure 8 shows the average accuracy results for four competitive methods. Our method kDAG achieves the best performance on all different k, which is slightly better than CAGG and much better than FEQ and LASP. This indicates an easy usability of kDAG in the SIGMOD'19 attractive topic recommendation.

8 RELATED WORK

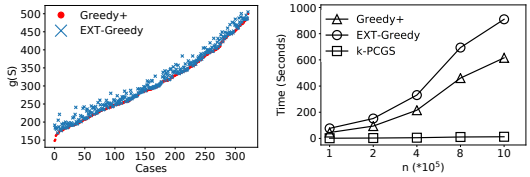
Our work is related to graph summarization and top-k diversification.

Graph summarization. In the literature, numerous works study the graph summarization [6, 11, 18, 26, 27, 27, 28]. A semi-structured data summarization approach is proposed for RDF graphs [6]. Gou et al. [11] propose a novel graph stream sketch to summarize the graph streams with linear space cost and constant update time cost. Both of these two works design data structures to store and summarize graphs. Kumar and Efstathopoulos [18] propose a framework to compute the utility for graph summarization and compress the graph with high utility. Different from the above studies focusing on the general graph structure, we focus on a summary optimization in DAGs with hierarchical relationships. Note that object summaries [8, 9] shares similar motivations, but has three major differences in data models, diversity definitions and problem formulations.



(a) ANIM (b) IMAGE (c) YAGO (d) UCI

Figure 6: Efficiency evaluation of Greedy+, EXT-Greedy, k-PCGS and CAGG methods on DAGs



(a) Approximation evaluation (b) Scalability test

Figure 7: Evaluation on synthetic DAGs.

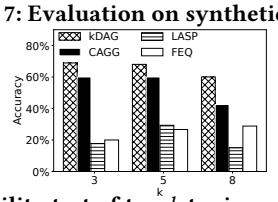


Figure 8: Usability test of top-k topic recommendation.

Top-k diversification. There exist several studies on the top-k graph diversification [5, 10, 12, 13, 21, 22, 29–31]. Qin et al.[21] investigate the diversified top-k search results in a graph. Ranu et al. [22] propose NB-Index to solve the top-k representative queries on graph databases. Based on a level-wise subgraph search, DSQL is proposed for top-k diversified subgraph querying in [29]. Long et al. [30] find top-k maximal cliques that can cover most number of nodes in a graph. Different from the above works on the top-k diversification on graph databases, subgraph queries, and cliques, this paper studies the graph summarization problem in hierarchical DAGs.

9 CONCLUSION

In this paper, we formulate and study a new kDAG-problem, which finds k representative vertices to summarize a hierarchical DAG associated with vertex weights. Due to the problem NP-hardness, we propose efficient greedy algorithms to tackle it. In addition, we develop two improved algorithms to find better answers with a theoretical guarantee in quality and be faster with theoretical complexity analysis, respectively. The k-PCGS method is scalable based on the candidate pruning and DAG compression. Extensive experiments validate the effectiveness and efficiency of our proposed algorithms.

ACKNOWLEDGEMENT

This paper is supported by NSFC 61702435, HK RGC GRF 12200917, 12201518, 12232716, 12201119, HK RGC CRF C6030-18G, and Guangdong Basic and Applied Basic Research Foundation (Project No. 2019B1515130001). Xin Huang is the corresponding author.

REFERENCES

[1] http://disease-ontology.org.
[2] https://www.snomed.org/snomed-ct/five-step-briefing.
[3] https://dl.acm.org/sig/sigmod.
[4] Leman Akoglu, Duen Horng Chau, U Kang, Danai Koutra, and Christos Faloutsos. 2012. Opavion: Mining and visualization in large graphs. In *SIGMOD*. 717–720.
[5] Ilio Catallo, Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. 2013. Top-k diversity queries over bounded regions. *TODS* 38, 2 (2013), 10.
[6] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. 2015. Query-oriented summarization of RDF graphs. *PVLDB* 8, 12 (2015), 2012–2015.
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. 248–255.
[8] Georgios Fakas, Zhi Cai, and Nikos Mamoulis. 2015. Diverse and proportional size-l object summaries for keyword search. In *SIGMOD*. 363–375.
[9] Georgios J Fakas, Zhi Cai, and Nikos Mamoulis. 2011. Size-l object summaries for relational keyword search. *PVLDB* 5, 3 (2011), 229–240.
[10] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Diversified top-k graph pattern matching. *PVLDB* 6, 13 (2013), 1510–1521.
[11] Xiangyang Gou, Lei Zou, Chenxingyu Zhao, and Tong Yang. 2019. Fast and Accurate Graph Stream Summarization. In *ICDE*. 1118–1129.
[12] Jinbin Huang, Xin Huang, Yuanyuan Zhu, and Jianliang Xu. 2019. Parameter-free Structural Diversity Search. In *WISE*. 677–693.
[13] Xin Huang, Hong Cheng, Rong-Hua Li, Lu Qin, and Jeffrey Xu Yu. 2013. Top-k structural diversity search in large networks. *PVLDB* 6, 13 (2013), 1618–1629.
[14] Xin Huang, Byron Choi, Jianliang Xu, William K Cheung, Yanchun Zhang, and Jiming Liu. 2017. Ontology-based Graph Visualization for Summarized View. In *CIKM*. 2115–2118.
[15] X Jing, JJ Cimino, et al. 2014. A Complementary Graphical Method for Reducing and Analyzing Large Data Sets. *Methods of information in medicine* 53, 3 (2014), 173–185.
[16] Xia Jing and James J Cimino. 2011. Graphical methods for reducing, visualizing and analyzing large data sets using hierarchical terminologies. In *AMIA Annual Symposium Proceedings*, Vol. 2011. 635.
[17] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. 85–103.
[18] K Ashwin Kumar and Petros Efstathopoulos. 2018. Utility-driven graph summarization. *PVLDB* 12, 4 (2018), 335–347.
[19] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* 407, 1-3 (2008), 458–473.
[20] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2013. Yago3: A knowledge base from multilingual wikipedias.
[21] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying top-k results. *PVLDB* 5, 11 (2012), 1124–1135.
[22] Sayan Ranu, Minh Hoang, and Ambuj Singh. 2014. Answering top-k representative queries on graph databases. In *SIGMOD*. 1163–1174.
[23] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293.
[24] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 1998. A metric for distributions with applications to image databases. In *ICCV*. 59–66.
[25] Yufei Tao, Yuanbing Li, and Guoliang Li. 2019. Interactive graph search. In *SIGMOD*. 1393–1410.
[26] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. 2008. Efficient aggregation for graph summarization. In *SIGMOD*. 567–580.
[27] You Wu, Junyang Gao, Pankaj K Agarwal, and Jun Yang. 2017. Finding diverse, high-value representatives on a surface of answers. *PVLDB* 10, 7 (2017), 793–804.
[28] X Yang, CM Procopiuc, and D Srivastava. 2011. Summary graphs for relational database schemas. *PVLDB* 4, 11 (2011), 899–910.
[29] Zhengwei Yang, Ada Wai-Chee Fu, and Ruifeng Liu. 2016. Diversified top-k subgraph querying in a large graph. In *SIGMOD*. 1167–1182.
[30] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2016. Diversified top-k clique search. *The VLDB Journal* 25, 2 (2016), 171–196.
[31] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. 2010. Solving the apparent diversity-accuracy dilemma of recommender systems. *PNAS* 107, 10 (2010), 4511–4515.
[32] Xuliang Zhu, Xin Huang, Byron Choi, Jianliang Xu, William K. Cheung, Yanchun Zhang, and Jiming Liu. 2020. Ontology-based Graph Visualization for Summarized View. arXiv:arXiv:2008.03053