Interactive Graph Search for Multiple Targets on DAGs

Zheng Wu[1], Xuliang Zhu[2], Yixiang Fang[3], Jianliang Xu[1], Xin Huang[1]
[1]Hong Kong Baptist University, China
[2] Antai College of Economics and Management, Shanghai Jiao Tong University, China
[3] The Chinese University of Hong Kong, Shenzhen, China
{cszhengwu, xujl, xinhuang}@comp.hkbu.edu.hk, zhu.xl@sjtu.edu.cn, fangyixiang@cuhk.edu.cn

ABSTRACT

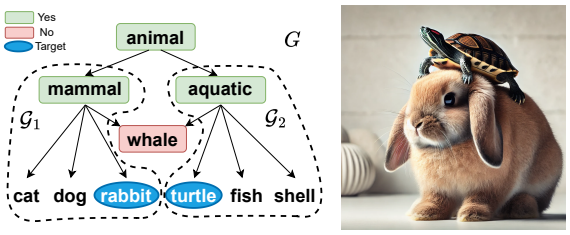
Interactive graph search (IGS) over DAGs aims to find a hidden target by asking interactive questions as few as possible. IGS is useful for many applications, e.g., facilitating supervised learning tasks by harnessing labeled data, image categorization, and product classification. However, most of the existing IGS methods only work for either *single target search on DAGs* or *multiple targets search on simple trees*. To overcome the gap, it motivates us to study a challenging and yet not solved problem of multiple targets search over DAGs. We analyze the new problem in-depth and propose a key concept of uncertain candidates. Based on it, we design an effective gain function to determine the best vertex to be asked questions and shrink the search space of potential targets greatly. Leveraging our uncertain candidates and gain function, we develop a unified k-EIS framework to search both single target and multiple targets. We analyze all algorithm complexities and theoretically show that our solution can significantly improve existing DFS-tree-based methods by asking $O(n)$ questions to $O(\log_2 n)$ questions in worst cases. To further improve IGS for multiple targets, we propose an advanced solution by dividing the whole DAG into k disjoint subgraphs with single targets and then tackling each subgraph one by one independently. Extensive experiments on real-world datasets validate that our proposed k-EIS framework can save lots of questions to search exact targets against four state-of-the-art IGS competitors.

PVLDB Reference Format:
Zheng Wu[1], Xuliang Zhu[2], Yixiang Fang[3], Jianliang Xu[1], Xin Huang[1]. Interactive Graph Search for Multiple Targets on DAGs. PVLDB, 14(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/i11ume/K-EIS.

1 INTRODUCTION

Crowdsourcing leverages human intelligence to address difficult tasks, which plays pivotal roles in many data-driven applications [6, 7, 10, 12, 21, 25, 32]. As one typical application of crowdsourcing, data labeling can provide high-quality labeled data, which are essentially important for numerous supervised learning tasks, e.g., image classification [5, 33], product categorization [19, 26], relational database search [2, 31], data filtering [27, 29, 34], cold-start recommendation [17, 19, 36], and so on. Given an unlabeled object, the task of interactive graph search (IGS) [31] leverages human intelligence to locate an exact target vertex for this object on a directed acyclic graph (DAG), where each vertex is a label and a directed edge represents the relationship of concept-of-instance. However, existing studies of interactive graph search [3, 19, 35] mainly focus on the search of *single target*, indicating that each object can have only one label, which is unsatisfactory in a lot of particular scenarios.



(a) DAG G with labels (b) Unlabeled image

Figure 1: An example of a hierarchical DAG in ImageNet (on the left) and an unlabeled image (on the right). The target labels of image are $\mathcal{T}$ ={"rabbit", "turtle"}.

In many real applications, an unlabeled object needs *multiple labels* as targets to accurately describe the object. For example, the image in Figure 1(b) has two kinds of animal, indicating that two labels of "rabbit" and "turtle" in the label DAG in Figure 1(a) are needed to depict this image. Similar cases of multiple targets are as follows. In academic research, a paper can have multiple keywords of ACM Computing Classification System as the target, which can be enough to accurately describe its research topic areas. In biomedicine, a new virus of "COVID-19" that leads to pneumonia, may have two labels of "virus" and "pneumonia" in the disease ontology. In recommendation systems, it needs to suggest interest labels for new users, where the user preferences may be diverse and not limited to a single interest, e.g., one may simultaneously like "music", "traveling", "sports", and so on. In this work, we study a new problem of interactive search for multiple targets over DAGs, which aims to find exact k targets using the minimum number of asking questions. Continuing the above example to find two labels for the image in Figure 1(b), we can ask a question in the form of "Is this an x?". Assume that the first question on x is "mammal", the answer is "Yes". Because there is a path from "mammal" to "rabbit", indicating a rabbit is a mammal. Next, we ask the second question on "whale", the answer is "No". Because there is no path from "mammal" to neither "rabbit" nor "turtle", indicating no whale

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

appears in the image. Finally, it seeks for at least six questions to exactly identify two final targets "rabbit" and "turtle".

In terms of data management perspective, it brings significant challenges to address our task of *interactive search for multiple targets*. The reasons are two-fold. First, the design of efficient search algorithms is vital, as a large number of unlabeled data is generated every day in real life. Especially, those user-generated data frequently happen on the web, e.g., user-uploaded images and videos, AI-generated articles, which are produced greatly in a streaming way. Second, the development of effective solutions is difficult but very important, which directly affects the budget cost of asked questions w.r.t. human resources in economic aspect. In addition, an unknown number of exact labels in a given data object also brings challenges for a crowdsourcing-based data labeling system, which is difficult to effectively handle. To tackle our problem, one straightforward idea is to extend existing single-target methods [3, 19, 35] and tree-based multi-target method KBM-IGS [36]. Unfortunately, they cannot work well for our problem, due to the reasons as follows.

(1) *Difficulty of multiple targets search in DAGs.* KBM-IGS [36] addressed the search for multiple targets in trees. However, in DAGs, vertices may have more than one parents and the structure is more dense. So applying KBM-IGS to DAGs is challenging due to the high time complexity involved. IGS [31] extracted a DFS-tree from DAGs and applied tree algorithms on the extracted tree. The DFS-tree-based method is correct for single target search. Because the unique target must be located within the subtree T_v following any question that receives a Yes answer, i.e., $\mathrm{Q}(v)$ = Yes. However, this method is not applicable for multiple targets, as multiple targets may reside outside the subtree T_v.

(2) *Limitation of DFS-tree in dense DAGs.* In tree-like DAGs, the DFS-tree method is effective as it retains most of the structural information after extracting a DFS-tree. However, in dense DAGs, the extraction of a DFS-tree removes many edges, leading to a significant loss of structural information. In bad cases, we theoretically show that DFS-tree-based methods require $O(n)$ questions to locate a single target.

(3) *Low effectiveness of exact multiple targets search.* KBM-IGS is designed to detect approximate targets in situations involving multiple targets. Although this method can be extended to detect exact targets, it needs to ask a large number of questions as validated in our experiments.

To tackle these bottlenecks, we propose a novel k-EIS framework for the exact search for multiple targets on DAGs. The key of our k-EIS framework includes two parts: *uncertain candidates* and *a new gain function* for selecting the best vertex to be asked. First, we give a new definition of uncertain candidates for those vertices that have an answer to be either yes or no, indicating an uncertain status. Thus, we can categorize all vertices into three statuses including yes, no, and also uncertain. This greatly helps us to refine candidate statues and shrink the search space of potential targets to exact ones, in contrast to *only two statuses* in the existing studies [36]. Second, we propose a new gain function based on the uncertain status to achieve a good balance trade-off by the worst case and average case. Built upon k-EIS, we design specific pruning rules to update candidate statuses, in terms of single target and multiple targets. To further improve the effectiveness of multiple targets search, we develop an improved algorithm called Mix-EIS. The key idea of Mix-EIS is divide-and-conquer. Generally speaking, it first divides the whole DAG into k disjoint subgraphs so that each subgraph has exactly one target. As the example of a DAG G with two targets shown in Figure 1, we ask three questions on "mammal", "whale" and "aquatic", receiving the answers "Yes", "No" and "Yes." After asking these three questions, the G is separated into two disjoint subgraphs, $\mathcal{G}_1$ and $\mathcal{G}_2$, so that each subgraph has a target. Finally, we independently use our k-EIS-based single target search algorithm to quickly identify each target on each subgraph, "rabbit" of $\mathcal{G}_1$, "turtle" of $\mathcal{G}_2$. To summarize, we make the following contributions:

Table 1: A detailed comparison of interactive search studies, including IGS, TS-IGS, BinG, KBM-IGS, and our k-EIS.

Methods	Target Size	Exact Answers	Data Domain
IGS [31]	Single	√	DAG
TS-IGS [35]	Single	√	DAG
BinG [4, 19]	Single	√	Tree
KBM-IGS [36]	Multiple	×	Tree
k-EIS (Ours)	Multiple	√	DAG

- We formulate a new problem of interactive graph search for identifying exact multiple targets on DAGs (Section 3).
- We propose a new concept of uncertain candidates, which contributes to an effective management of three typed candidates and potential targets over a series of dynamic question-and-answering. We design a gain function to ask questions on the best vertex for reducing uncertain candidates. Based on the above, we develop an effective k-EIS framework (Section 4).
- We develop updating rules and k-EIS-based algorithm for single target search. We analyze the algorithm complexities (Section 5).
- In the k-EIS framework, we first develop a k-EIS-based algorithm for multiple targets search. To further improve the effectiveness, we proposed a divide-and-conquer solution based on the partition of k disjoint subgraphs such that it finds single target within each subgraph (Section 6).
- We theoretically analyze the performance of our methods and existing methods on various shapes of DAGs. It shows that our methods outperform the existing DFS-tree-based methods in a few cases, which improves to asking from $O(n)$ questions to $O(\log_2 n)$ questions (Section 7).
- We conduct extensive experiments on real-world datasets against four state-of-the-art methods. The results show that our methods can use nearly 6% and 50% question cost for searching single target and multiple targets, respectively, in contrast to existing competitors (Section 8).

We review and compare existing studies in Section 2. Finally, we conclude the paper in Section 9.

2 RELATED WORK

We summarize and compare the four most relevant studies to our work, including IGS [31], TS-IGS [35], BinG [4, 19], and KBM-IGS [36], as shown in Table 1. Specifically, Tao et al. [31] advanced

the Interactive Graph Search (IGS) problem that dynamically adapted to responses to facilitate efficient target identification within Directed Acyclic Graphs (DAGs). The general idea was to extract a heavy DFS-tree and apply binary search and heavy-path decomposition. Building upon the IGS, Zhao et al. [35] further improved the IGS problem with the TS-IGS algorithm, which optimized the number of questions needed when targets were located at shallower depths within the graph. Furthermore, BinG [4, 19] was a greedy-based method that queried the vertex with maximal gain. It modeled the single-target problem as a decision tree construction problem and proved the approximate guarantee in theory. Expanding the application to multi-target scenario, Zhu et al. [36] developed the KBM-IGS method, the first to handle multiple targets within tree structures. It was a heuristic framework that provided a trade-off between target probability and benefit gain. This framework incorporated budget constraints to effectively approximate the target set under limited question budgets. *In summary, these three methods IGS [31], TS-IGS [35] and BinG [4, 19] only tackled single-target interactive search, which was inapplicable for multiple targets scenario. On the other hand, KBM-IGS [36] can detect approximate answers of multiple targets in trees, but is hard to search exact targets in DAGs.*

There were also several other graph search problems [3, 4, 8, 13–15, 18–20, 22, 23, 28, 30] related to our work. Human-assisted graph search [28] focused on optimizing the cost of human computations necessary for tasks in crowdsourcing services. It was an offline problem that asked all questions in one go. Cong et al. [4] tackled minimizing expected question numbers, assuming that objects followed a probabilistic distribution. However, obtaining the exact probabilistic distribution was difficult in the real world. Li et al. [19, 20] introduced multiple interactive graph search, which utilized a multiple-choice question to enhance the querying process. In the work [3], the authors explored the complexities added by human errors in responses within the IGS framework, focusing on the effect of inaccurate answers on the search process. Lu et al. [22, 23, 30] studied the partial order multiway search that asked k questions in each turn. In the single question setting, the method is the same as the IGS [31] method. *In comparison, all these works had different research focuses and problem settings from our* k-EIS *problem.*

3 PRELIMINARIES

3.1 Directed Acyclic Graph

Let $G = (V, E)$ be a directed acyclic graph (DAG) consisting of a set of vertices V and a set of directed edges E. Let $n = |V|$ and $m = |E|$ denote the size of vertices and edges, respectively. For a directed edge $\langle v, u\rangle \in E$, we say u is an in-neighbor of v, and v is an out-neighbor of u. For the set of out-neighbors, denote by $\mathsf{N}^+(v) = \{u \mid \langle v, u\rangle \in E\}$, and for the set of in-neighbors, denoted by $\mathsf{N}^-(v) = \{u \mid \langle u, v\rangle \in E\}$. Assume that there is a unique root r of G, i.e., $\mathsf{N}^-(r) = 0$ [31]. If there are multiple vertices with an in-degree of 0, we add a virtual vertex r with outgoing edges to each of these vertices.

Given two vertices u and v, we say that u can reach v (denoted by $u \rightarrow v$) if and only if there exists a directed path from u to v in G. Note that for any vertex v, it can reach itself, i.e., $v \rightarrow v$. If u cannot reach v, we denote it by $u \nrightarrow v$. In addition, the ancestors of a vertex

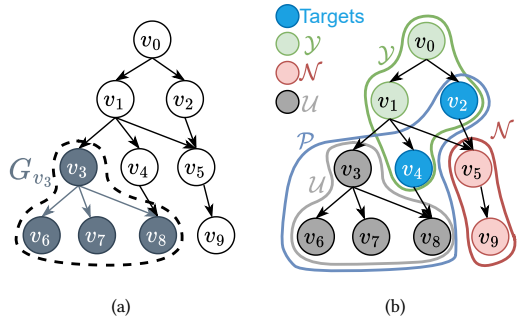


Figure 2: An example of a DAG G. In Figure 2(b), G has the hidden targets $\mathcal{T} = \{v_2, v_4\}$, and we have asked the questions $\mathcal{Q} = \{\mathrm{Q}(v_2) = \mathrm{Yes}, \mathrm{Q}(v_4) = \mathrm{Yes}, \mathrm{Q}(v_5) = \mathrm{No}\}$.

v, denoted by $\mathrm{anc}(v)$, are defined as the set of vertices that can reach v in G, i.e., $\mathrm{anc}(v) = \{u \in V \mid u \rightarrow v\}$. Similarly, the descendants of a vertex v, denoted by $\mathrm{des}(v)$, are defined as the set of vertices that are reachable from v in G, i.e., $\mathrm{des}(v) = \{u \in V \mid v \rightarrow u\}$. Furthermore, we define G_u as the subgraph consisting of all vertices and edges reachable from a vertex $u \in V$. Formally, $G_u = (V_u, E_u)$, where $V_u = \mathrm{des}(u)$ and $E_u = \{\langle v, w\rangle \in E \mid v, w \in \mathrm{des}(u)\}$.

Example 1. *Figure 2(a) shows an example of a DAG G rooted by $r = v_0$. We have $\mathsf{N}^+(v_3) = \{v_6, v_7, v_8\}$, $\mathsf{N}^-(v_3) = \{v_1\}$, $\mathrm{anc}(v_3) = \{v_0, v_1, v_3\}$, $\mathrm{des}(v_3) = \{v_3, v_6, v_7, v_8\}$. The subgraph G_{v_3} is formed by four vertices $\{v_3, v_6, v_7, v_8\}$ and three edges $\{\langle v_3, v_6\rangle, \langle v_3, v_7\rangle, \langle v_3, v_8\rangle\}$.*

3.2 Problem Formulation

In the following, we present the scheme of interactive graph search for multiple targets over DAGs. We start with three important concepts: *targets*, *interactive questions*, and *potential targets*.

Targets. The targets, denoted by $\mathcal{T} \subseteq V$, are a set of "hidden" vertices that we aim to identify in the DAG G. The number of targets is represented by $|\mathcal{T}| = k$, where $k \geq 1$. For $k = 1$, we say that $\mathcal{T}$ is a *single target*; for $k > 1$, $\mathcal{T}$ are *multiple targets*. We assume that the targets satisfy the independent condition, i.e., for any two distinct vertices $u, v \in \mathcal{T}$, there is no directed path from v to u, i.e., $\{u, v \in \mathcal{T} : \nexists v \nrightarrow u\}$.

Interactive questions. Given the targets $\mathcal{T}$ are unknown in advance, our system could interact with users by asking questions to identify these hidden targets $\mathcal{T}$. For a vertex $v \in V$, an interactive question is represented as $\mathrm{Q}(x)$, asking whether there exists a path from x to at least one of targets $u \in \mathcal{T}$, i.e., $\mathrm{Q}(x) = \mathrm{boolean}(x \rightarrow \mathcal{T})$. The question answer is "Yes" if one directed path exists from x to one target in $\mathcal{T}$, and "No" otherwise. We can ask a series of t questions in order, denoted by,

$$\mathcal{Q} = \{\mathrm{Q}(x_1), \mathrm{Q}(x_2), \ldots, \mathrm{Q}(x_t)\},$$

where x_i represents the vertex asked in the i-th question. Note that we assume that each question is answered correctly by users following [19, 28, 31, 36]. If users make wrong answers, we can adopt a few additional measures of quality control to improve the final accuracy of search targets, such as the majority voting, expert review, group consensus, and so on. For example, we can ask multiple users on the same question to seek additional answers, and determine the final answer based on the majority voting. We have conducted Exp-4 to validate the effectiveness in Section 8.

Potential targets. Following the definition in KBM-IGS [36], potential targets (called candidate set in [28, 31]), denoted by $\mathcal{P}$, are defined as a candidate set of vertices that might precisely be the targets, where $\mathcal{T} \subseteq \mathcal{P}$. Before asking any question, the whole set of vertices V is regarded as the potential targets, denoted as $\mathcal{P} = V$. In other words, each vertex $v \in \mathcal{P}$ could be a potential answer of targets in $\mathcal{T}$.

Based on the above concepts, we are ready to formulate the problem of interactive graph search for identifying exact k targets over DAGs (k-EIS) as follows.

Problem 1 (k-EIS Problem). *Given a DAG $G = (V, E)$, hidden targets $\mathcal{T} \subseteq V$, potential targets $\mathcal{P} = V$, and a positive integer $k \in \mathbb{Z}^+$ where $k = |\mathcal{T}|$, the objective of* k-EIS *problem is to interactively refine potential answers from $\mathcal{P}$ to $\mathcal{P}^* \subseteq \mathcal{P}$ via asking a series of t questions $\mathcal{Q} = \{\mathrm{Q}(x_1), \mathrm{Q}(x_2), \ldots, \mathrm{Q}(x_t)\}$, such that it achieves the finalized $\mathcal{P}^* = \mathcal{T}$ using the minimum number of t questions.*

Example 2. *Assume that the targets $\mathcal{T} = \{v_2, v_4\}$ and $k = 2$ in the DAG G shown in Figure 2(b), we have asked the questions $\mathcal{Q} = \{\mathrm{Q}(v_2) = \text{Yes}, \mathrm{Q}(v_4) = \text{Yes}, \mathrm{Q}(v_5) = \text{No}\}$ and the potential targets $\mathcal{P} = \{v_2, v_3, v_4, v_6, v_7, v_8\}$. With one question asked on v_3, the potential targets $\mathcal{P}$ is finalized as $\mathcal{P}^* = \{v_2, v_4\}$, where $\mathcal{P}^* = \mathcal{T}$.*

4 THE PROPOSED FRAMEWORK

This section introduces our proposed k-EIS framework to find exact k targets in a DAG with a few questions. The key is to carefully select a vertex with the most worth for asking questions to shrink potential targets $\mathcal{P}$ greatly. To this end, we first leverage the previous questions' answers to categorize all vertices into three statuses: Yes-candidates ($\mathcal{Y}$), No-candidates ($\mathcal{N}$), and Uncertain-candidates ($\mathcal{U}$), which can prune disqualified candidates and help identify potential targets. Second, we give a new gain function based on the proposed three statuses to select the best vertex for asking questions.

4.1 Dynamic Candidate Management

We start by defining all candidate vertices into three statuses: Yes-candidates ($\mathcal{Y}$), No-candidates ($\mathcal{N}$), and Uncertain-candidates ($\mathcal{U}$). Different from the existing study of multi-target search over trees, i.e. KBM-IGS [36], that only uses two statuses $\mathcal{Y}$ and $\mathcal{N}$, we have a new statue of Uncertain-candidates ($\mathcal{U}$), which are very useful for candidate pruning.

- **Yes-candidates ($\mathcal{Y}$):** The Yes-candidates is defined as the set of vertices confirmed to reach at least one target, denoted by $\mathcal{Y} = \{v \in V \mid \mathrm{Q}(v) = \text{Yes}\}$.
- **No-candidates ($\mathcal{N}$):** The No-candidates comprises all vertices that are determined unable to reach any targets, represented as $\mathcal{N} = \{v \in V \mid \mathrm{Q}(v) = \text{No}\}$.
- **Uncertain-candidates ($\mathcal{U}$):** The Uncertain-candidates is the set of vertices that are not determined whether they can reach targets or not. The answer could be either yes or no. We denoted by $\mathcal{U} = V \setminus \{\mathcal{Y} \cup \mathcal{N}\}$. For initialization, we have $\mathcal{U} = V$. With more questions $\mathcal{Q}$ are asked, $\mathcal{U}$ continuously refines with more information of $\mathcal{Y}$ and $\mathcal{N}$.

At each round, it is worth asking questions on an uncertain candidate vertex $x \in \mathcal{U}$, but not the deterministic one of $\mathcal{Y}$ and $\mathcal{N}$. For each question $\mathrm{Q}(x)$, we have two important rules in the following.

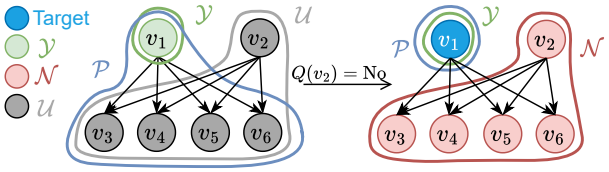


Figure 3: An example of a DAG G with a hidden target v_1. After asking the question on the Uncertain-candidate v_2, $\mathrm{Q}(v_2) =$ No, the target v_1 can be found.

Rule 1. *If a vertex $x \in \mathcal{U}$, and $\mathrm{Q}(x) =$ Yes, then it is conclusively any vertex $y \in \mathrm{anc}(x)$, $\mathrm{Q}(y) =$ Yes.*

Proof. Suppose that $\mathrm{Q}(x) =$ Yes for $x \in \mathcal{U}$. This indicates $x \to v$ for some $v \in \mathcal{T}$. For any $y \in \mathrm{anc}(x)$, $y \to x$. Therefore, the path $y \to x \to v$ exists, thus $\mathrm{Q}(y) =$ Yes holds. □

Rule 2. *If a vertex $x \in \mathcal{U}$, and $\mathrm{Q}(x) =$ No, then it is conclusively any vertex $y \in \mathrm{des}(x)$, $\mathrm{Q}(y) =$ No.*

Proof. Suppose that $\mathrm{Q}(x) =$ No for $x \in \mathcal{U}$. This indicates $x \nrightarrow v$ for any $v \in \mathcal{T}$. For any $y \in \mathrm{des}(x)$, $x \to y$. If $\mathrm{Q}(y) =$ Yes holds, there exists a path $x \to y \to v$, $v \in \mathcal{T}$, contradicting $\mathrm{Q}(x) =$ No. Thus, $\mathrm{Q}(y) =$ No holds. □

Based on $\mathcal{Y}$ and $\mathcal{U}$, we infer the potential targets $\mathcal{P}$ as follows.

Lemma 4.1. *The potential targets $\mathcal{P} \subseteq \mathcal{Y}^* \cup \mathcal{U}$, where $\mathcal{Y}^* = \{u \in \mathcal{Y} : \nexists w \in \mathcal{Y}$ such that $u \to w\}$. Here, $\mathcal{P} \subseteq \mathcal{Y} \cup \mathcal{U}$.*

Proof. Assume that $\mathcal{P} \cap \mathcal{N} = \{x\}$. $x \in \mathcal{N}$ indicates that $x \nrightarrow u$ for any $u \in \mathcal{T}$, so $x \notin \mathcal{P}$, contradicting $x \in \mathcal{P}$. Thus, $\mathcal{P} \cap \mathcal{N} = \emptyset$ with $V = \mathcal{Y} \cup \mathcal{N} \cup \mathcal{U}$, $\mathcal{P} \subseteq \mathcal{Y} \cup \mathcal{U}$ holds. Suppose that a vertex $v \in \mathcal{Y}, v \notin \mathcal{Y}^*$ is a target, $v \notin \mathcal{Y}^*$ indicates that $\exists u \mid v \to u, u \in \mathcal{Y}$, $u \in \mathcal{Y}$ indicates that there exists a path $v \to u \to w, w \in \mathcal{T}$, contracting the definition of targets $\{v, w \in \mathcal{T} : \nexists v \nrightarrow w\}$. Thus, for any vertex $v \in \mathcal{Y}, v \notin \mathcal{Y}^*$ cannot be a target. So, $\mathcal{P} \subseteq \mathcal{Y}^* \cup \mathcal{U}$ holds. □

As a result, to find exact targets $\mathcal{T}$, we need to shrink $\mathcal{Y}$ and $\mathcal{U}$ to make the potential targets $\mathcal{P}$ close to $\mathcal{T}$.

Example 3. *Figure 2(b) illustrates the hidden targets $\mathcal{T} = \{v_2, v_4\}$ within a DAG G. Suppose that we have asked questions $\mathcal{Q} = \{\mathrm{Q}(v_2) =$ Yes, $\mathrm{Q}(v_4) =$ Yes, $\mathrm{Q}(v_5) =$ No$\}$. Consequently, according to Rule 1, vertices v_0, v_1, v_2 and v_4 are classified into Yes-candidates, forming the Yes-candidates $\mathcal{Y} = \{v_0, v_1, v_2, v_4\}$. According to Rule 2, vertices v_5, and v_9 are classified into No-candidates, hence the No-candidates $\mathcal{N} = \{v_5, v_9\}$. The vertices v_3, v_6, v_7, and v_8 remain in the Uncertain-candidates, denoted $\mathcal{U} = \{v_3, v_6, v_7, v_8\}$. The potential targets, $\mathcal{P}$, includes vertices $\{v_2, v_3, v_4, v_6, v_7, v_8\}$. When we ask one more question on v_3, v_3 has no path to any target v_2 or v_4, thus $\mathrm{Q}(v_3) =$ No, according to the Rule 2, $\{v_3, v_6, v_7, v_8\}$ would be classified into $\mathcal{N}$ from $\mathcal{U}$. Currently, $\mathcal{U}$ is an empty set, the potential targets $\mathcal{P} = \{v_2, v_4\}$ finalized $\mathcal{P} = \mathcal{T}$.*

Benefit of Uncertain-candidates ($\mathcal{U}$). We use the newly proposed Uncertain-candidates, which can effectively reduce the search space for finding targets fast. Different from the vertex of potential targets, the vertex of Uncertain-candidates could be not a target.

Thus, we can ask a vertex x of Uncertain-candidates, but x is not a potential target, i.e., $x \in \mathcal{U} \setminus \mathcal{P}$. The following example in Figure 3 illustrates the benefit of Uncertain-candidates against potential targets. Our method using Uncertain-candidates can locate the target by only **one** question, instead of **four** questions of other methods using potential targets.

Example 4. *Figure 3 shows an example of a DAG G with a single hidden target $\mathcal{T} = \{v_1\}$. We have $\mathcal{U} = \{v_2, v_3, v_4, v_5, v_6\}$, $\mathcal{P} = \{v_1, v_3, v_4, v_5, v_6\}$ and $\mathcal{Y} = \{v_1\}$. As $\mathcal{U} \neq \mathcal{P}$, we ask one question on the Uncertain-candidate $v_2 \in \mathcal{U} \setminus \mathcal{P}$. Since v_2 cannot reach v_1, we find that $\mathrm{Q}(v_2) = \mathrm{No}$. According to Rule 2, $\{v_3, v_4, v_5, v_6\}$ are No-candidates. Consequently, v_1 can be identified as the target. If a search algorithm only focuses on potential targets, then v_2 cannot be asked for questions, reflecting that four additional questions $\{\mathrm{Q}(v_3) = \mathrm{No}, \mathrm{Q}(v_4) = \mathrm{No}, \mathrm{Q}(v_5) = \mathrm{No}, \mathrm{Q}(v_6) = \mathrm{No}\}$ are needed to locate the target v_1.*

4.2 Gain Calculation

Leveraging three kinds of candidates $\mathcal{Y}$, $\mathcal{N}$, and $\mathcal{U}$, we give a novel definition of $\mathsf{gain}(x)$ to evaluate the goodness of asking question on vertex x, i.e., $\mathrm{Q}(x)$. We consider the answer of $\mathrm{Q}(x)$ into two cases.

- If $\mathrm{Q}(x) = \mathrm{Yes}$, a few candidates $u \in \mathcal{U}$ change from $\mathcal{U}$ to $\mathcal{Y}$ or $\mathcal{N}$. The change from $\mathcal{U}$ to $\mathcal{Y}$ can be inferred by Rule 1. Note that the change from $\mathcal{U}$ to $\mathcal{N}$ may happen for $|\mathcal{T}| = 1$. Assume that the updated $\mathcal{U}$ is $\hat{\mathcal{U}}_Y$. Thus, we denote the number of such updated vertices by $\mathsf{cnt}_\mathsf{Y}(x)$, i.e., $\mathsf{cnt}_\mathsf{Y}(x) = |\mathcal{U}| - |\hat{\mathcal{U}}_Y|$.
- If $\mathrm{Q}(x) = \mathrm{No}$, a few candidates $u \in \mathcal{U}$ change from $\mathcal{U}$ to $\mathcal{N}$ or $\mathcal{Y}$. The change from $\mathcal{U}$ to $\mathcal{N}$ can be inferred by Rule 2. Note that the change from $\mathcal{U}$ to $\mathcal{Y}$ may happen for $|\mathcal{T}| = 1$. Assume that the updated $\mathcal{U}$ is $\hat{\mathcal{U}}_N$. Thus, we denote the number of such updated vertices by $\mathsf{cnt}_\mathsf{N}(x)$, i.e., $\mathsf{cnt}_\mathsf{N}(x) = |\mathcal{U}| - |\hat{\mathcal{U}}_N|$.

Integrating the above two cases in a balanced way, $\mathsf{gain}(x)$ is defined as follows:

$$\mathsf{gain}(x) = \mathsf{cnt}_\mathsf{Y}(x) \times \mathsf{cnt}_\mathsf{N}(x). \tag{1}$$

Our function $\mathsf{gain}(x)$ can achieve a good balance of the reductions of Uncertain-candidates ($\mathcal{U}$) w.r.t. two question answers. Alternatively, one may propose to use another generalized gain function in terms of $\mathsf{gain}(x) = f(\mathsf{cnt}_\mathsf{Y}(x), \mathsf{cnt}_\mathsf{N}(x))$. Here, the function $f(.)$ needs to achieve three good properties: *symmetry*, *non-decreasing gain*, and *risk-aware balance*. First, $f(.)$ is symmetry, i.e., $f(a, b) = f(b, a)$. Second, $f(a_1, b) \geq f(a_2, b)$ if and only if $a_1 \geq a_2 \geq 0$, implying the non-decreasing gain. Third, f should make a good trade-off balancing the benefits of two different answers $\mathsf{cnt}_\mathsf{Y}(x)$ and $\mathsf{cnt}_\mathsf{N}(x)$ as either case may incur. Our $\mathsf{gain}(x)$ in Eq. 1 successfully admits the above three properties. Other choices, e.g., the minimum $f(a, b) = \min(a, b)$ and the harmonic mean $f(a, b) = \frac{2ab}{a+b}$, cannot be better than ours. In summary, this justifies the design choice of our gain function $\mathsf{gain}(x)$.

Algorithm 1 k-EIS Framework

Input: DAG $G = (V, E)$, an integer k.
Output: Targets $\mathcal{T}$ with $|\mathcal{T}| = k$.

1: Let $\mathcal{Y} \leftarrow \{r\}$, $\mathcal{N} \leftarrow \emptyset$,$\mathcal{U} \leftarrow \{V\} \setminus \{r\}$, $\mathcal{P} \leftarrow \{V\} \setminus \{r\}$;
2: **while** $\mathcal{U} \neq \emptyset$ **do**
3: **for** each uncertain candidate $v \in \mathcal{U}$ **do**
4: Calculate the reduced number of uncertain candidates $\mathsf{cnt}_\mathsf{Y}(v)$ by assuming that $\mathrm{Q}(v) = \mathrm{Yes}$;
//Note that we calculate $\mathsf{cnt}_\mathsf{Y}(v)$ by invoking Algo. 2 for single target with $k = 1$ and Algo. 3 for multiple targets with $k > 1$, respectively. No $\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}$ are actually updated during this process. Likewise, similar actions are performed for the following calculating $\mathsf{cnt}_\mathsf{N}(v)$.
5: Calculate $\mathsf{cnt}_\mathsf{N}(v)$ for $\mathrm{Q}(v) = \mathrm{No}$ by invoking Algo. 2 or 3;
6: Calculate $\mathsf{gain}(v) \leftarrow \mathsf{cnt}_\mathsf{Y}(v) \times \mathsf{cnt}_\mathsf{N}(v)$; ▷ by Eq. 1
7: Select $v^* \leftarrow \arg\max_{v \in \mathcal{U}} \mathsf{gain}(v)$ and ask the question $\mathrm{Q}(v^*)$;
8: **if** $\mathrm{Q}(v^*) = \mathrm{Yes}$ **then**
9: Update potential targets and three candidate sets $\{\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}\}$ by invoking Algo. 2 or 3 for $\mathrm{Q}(v^*) = \mathrm{Yes}$;
10: **else**
11: Update potential targets and three candidate sets $\{\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}\}$ by invoking Algo. 2 or 3 for $\mathrm{Q}(v^*) = \mathrm{No}$;
12: **return** $\mathcal{P}$ as the final targets;

4.3 k-EIS Framework

Based on gain evaluation for reducing uncertain candidates, We proposed k-EIS framework for multiple targets search. The key idea of k-EIS is to greedily select a vertex $v^* \in \mathcal{U}$ with the largest $\mathsf{gain}(v^*)$ for asking question $\mathrm{Q}(v^*)$ and then update $\mathcal{Y}, \mathcal{N}, \mathcal{U}$ at each round until all candidates are deterministic, i.e., $\mathcal{U} = \emptyset$.

The algorithm of k-EIS framework is detailed in Algo. 1. The framework can handle two search cases for single target with $|\mathcal{T}| = k = 1$ and multiple targets with $|\mathcal{T}| = k > 1$. We deal with these two cases by developing different gain calculation procedures in Algo. 2 and Algo. 3, respectively. Initially, it creates four different subsets from V (line 1): $\mathcal{Y} \leftarrow \{r\}$, which contains the root and denotes Yes-candidates; $\mathcal{N} \leftarrow \emptyset$, representing No-candidates; $\mathcal{U} \leftarrow \{V\} \setminus \{r\}$, comprising all other vertices as the Uncertain-candidates; and $\mathcal{P} \leftarrow \{V\} \setminus \{r\}$, indicating potential targets. Before asking each question, the algorithm calculates the cnt_Y and cnt_N of each vertex v in the Uncertain-candidates $\mathcal{U}$ (lines 3-6). It calculates the number of vertices that will transition from $\mathcal{U}$ to other subsets($\mathcal{Y}$ or $\mathcal{N}$) if $\mathrm{Q}(v) = \mathrm{Yes}$, which determines $\mathsf{cnt}_\mathsf{Y}(v)$ (line 4). Similarly, assuming $\mathrm{Q}(v) = \mathrm{No}$, it computes $\mathsf{cnt}_\mathsf{N}(v)$ (line 5). The gain of each vertex is then calculated using the function $\mathsf{gain}(v) = \mathsf{cnt}_\mathsf{Y}(v) \times \mathsf{cnt}_\mathsf{N}(v)$ (line 6). Following this, the vertex v^* with the largest gain is selected and queried (line 7). After asking the question $\mathrm{Q}(v^*)$, the subsets $\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}$ are updated by SingleEvalSync or MultipleEvalSync with a specific answer "Yes" or "No" (lines 8-11). Finally, the algorithm terminates until $\mathcal{U} = \emptyset$ and returns the answer $\mathcal{P}$ for the identified targets (line 12).

5 SINGLE TARGET SEARCH

In this section, we develop k-EIS algorithm for single target search based on our framework in Algo. 1, where $|\mathcal{T}| = k = 1$.

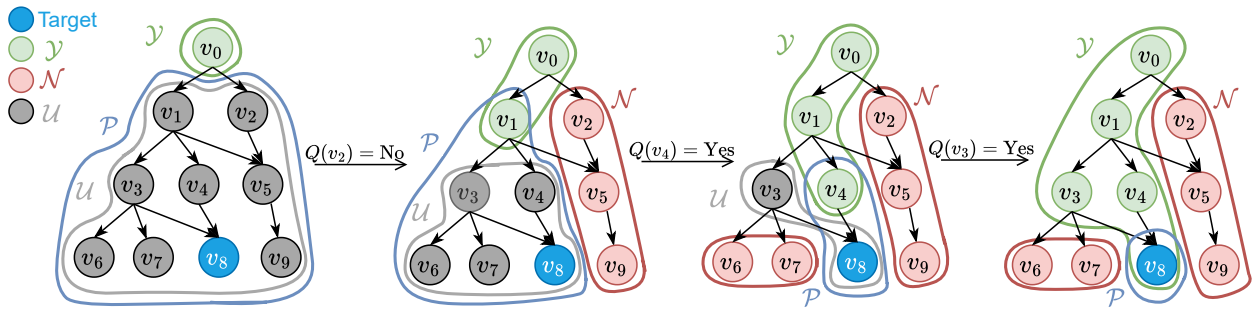


Figure 4: The k-EIS algorithm finds a hidden target v_8 on the DAG G rooted by $r = v_0$, showing questions and updates of candidates. And it asks three questions $Q = \{\mathrm{Q}(v_2) = \text{No}, \mathrm{Q}(v_4) = \text{Yes}, \mathrm{Q}(v_3) = \text{Yes}\}$ to find the exact target v_8.

Algorithm 2 SingleEvalSync

Input: Queried vertex x with $\mathrm{Q}(x)$, $\mathcal{P}$, $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$.
Output: $\mathrm{cnt}(x)$, $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, $\hat{\mathcal{U}}$

1: $\hat{\mathcal{P}} \leftarrow \mathcal{P}$, $\hat{\mathcal{Y}} \leftarrow \mathcal{Y}$, $\hat{\mathcal{N}} \leftarrow \mathcal{N}$, $\hat{\mathcal{U}} \leftarrow \mathcal{U}$;
2: **if** $\mathrm{Q}(x) = \text{Yes}$ **then**
3: $\quad \mathcal{R} \leftarrow \mathrm{anc}(x) \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{Y}} \leftarrow \hat{\mathcal{Y}} \cup \mathcal{R}$; ▷ By Rule 1
4: $\quad \hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cap \mathrm{des}(x)$;
5: **else**
6: $\quad \mathcal{R} \leftarrow \mathrm{des}(x) \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{N}} \leftarrow \hat{\mathcal{N}} \cup \mathcal{R}$; ▷ By Rule 2
7: $\quad \hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cap \{V \setminus \mathrm{des}(x)\}$;
8: Let y be the vertex which has the minimal topological order in $\mathcal{P}$;
9: **if** y can reach all the targets $\mathcal{P}$ **then**
10: $\quad \mathcal{R} \leftarrow \mathrm{anc}(y) \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{Y}} \leftarrow \hat{\mathcal{Y}} \cup \mathcal{R}$;
▷ By Rule 1 and Rule 3 (lines 9-11)
11: $\mathcal{R} \leftarrow V \setminus \bigcup_{v \in \hat{\mathcal{P}}} \mathrm{anc}(v)$;
12: $\mathcal{R} \leftarrow \mathcal{R} \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{N}} \leftarrow \hat{\mathcal{N}} \cup \mathcal{R}$; ▷ By Rule 4
13: $\mathrm{cnt}(x) \leftarrow |\mathcal{U}| - |\hat{\mathcal{U}}|$;
14: **return** $\mathrm{cnt}(x)$, $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, $\hat{\mathcal{U}}$;

5.1 Candidate Updating Rules for Single Target

In the scenario of single target, the potential targets $\mathcal{P}$ plays a pivotal role, which refines the target's scope based on the asked questions.

Updating $\mathcal{P}$. For asking a question on vertex $v \in \mathcal{U}$, $\mathcal{P}$ will be updated in these two conditions.

- If $\mathrm{Q}(v) = \text{Yes}$, then $\mathcal{P}$ is intersected with $\mathrm{des}(v)$, narrowing down the potential targets to the descendants of v, i.e., if $\mathrm{Q}(v) = \text{Yes}$, then $\mathcal{P} \leftarrow \mathcal{P} \cap \mathrm{des}(v)$.
- Otherwise, if $\mathrm{Q}(v) = \text{No}$, $\mathcal{P}$ is updated to exclude $\mathrm{des}(v)$, refining the potential targets outside the descendants of v, i.e., if $\mathrm{Q}(v) = \text{No}$, then $\mathcal{P} \leftarrow \mathcal{P} \cap \{V \setminus \mathrm{des}(v)\}$.

Updating $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$. After the updated $\mathcal{P}$, we can update other candidates $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$ via the following useful rules, accordingly.

Rule 3. *For a vertex $v \in \mathcal{U}$ with $\mathrm{des}(v) \cap \mathcal{P} = \mathcal{P}$, we determine that $\mathrm{Q}(v) = \text{Yes}$, $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{v\}$ and $\mathcal{U} \leftarrow \mathcal{U} \setminus \{v\}$.*

Proof. Suppose that $\mathrm{des}(v) \cap \mathcal{P} = \mathcal{P}$ for $v \in \mathcal{U}$. This indicates $v \to u$ for all $u \in \mathcal{P}$. Since $\mathcal{T} \subseteq \mathcal{P}$, $v \to t$ for every $t \in \mathcal{T}$. Thus, $\mathrm{Q}(v) = \text{Yes}$ holds. □

Rule 4. *For a vertex $v \in \mathcal{U}$ with $\mathrm{des}(v) \cap \mathcal{P} = \emptyset$, we determine that $\mathrm{Q}(v) = \text{No}$, $\mathcal{N} \leftarrow \mathcal{N} \cup \{v\}$ and $\mathcal{U} \leftarrow \mathcal{U} \setminus \{v\}$.*

Proof. Suppose that $\mathrm{des}(v) \cap \mathcal{P} = \emptyset$ for $v \in \mathcal{U}$. This indicates $v \not\to u$ for all $u \in \mathcal{P}$. Since $\mathcal{T} \subseteq \mathcal{P}$, $v \not\to t$ for any $t \in \mathcal{T}$. Thus, $\mathrm{Q}(v) = \text{No}$ holds. □

These rules can also help further update candidates, combining with Rule 1 and Rule 2 in Section 4.1.

5.2 Single Target Search Algorithm

We propose the k-EIS algorithm for single target search.

k-EIS algorithm for single target. The k-EIS algorithm combines the k-EIS framework and algorithm SingleEvalSync for evaluating the gain of each vertex and updating three sets of candidates ($\mathcal{Y}, \mathcal{N}, \mathcal{U}$) and potential targets ($\mathcal{P}$). We introduce the SingleEvalSync in detail in Algo. 2. Initially, the algorithm duplicates the sets $\mathcal{P}$, $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$ into temporary sets $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, and $\hat{\mathcal{U}}$, ensuring no direct modification to the original sets (line 1). If $\mathrm{Q}(x) = \text{Yes}$, according to Rule 1, it performs the following updates (lines 2-4): Identifies all vertices in $\hat{\mathcal{U}}$ that can reach the queried vertex x, i.e., $\mathrm{anc}(x) \cap \hat{\mathcal{U}}$, and classifies these vertices into $\hat{\mathcal{Y}}$, updating $\hat{\mathcal{U}}$ to exclude these vertices (line 3). Narrows down the potential targets $\hat{\mathcal{P}}$ to only include the descendants of x, as these are the only vertices that could be the targets (line 4). Otherwise, if $\mathrm{Q}(x) = \text{No}$, according to Rule 2, it performs the following updates (lines 5-7): Identifies all vertices in $\hat{\mathcal{U}}$ that the queried vertex x can reach, i.e., $\mathrm{des}(x) \cap \hat{\mathcal{U}}$, and classifies these vertices into $\hat{\mathcal{N}}$, updating $\hat{\mathcal{U}}$ to exclude these vertices (line 6). It updates $\hat{\mathcal{P}}$ to exclude all descendants of x, as none of these vertices can be targets (line 7). Additionally, if a vertex y which has the minimal topological order within $\hat{\mathcal{P}}$ can reach all the potential targets $\hat{\mathcal{P}}$, it is definitively classified as a Yes-candidate, per Rule 3 (lines 8-10). With the $\mathrm{Q}(y) = \text{Yes}$, according to the Rule 1, the ancestors of y in $\hat{\mathcal{U}}$ could be classified into $\hat{\mathcal{Y}}$ from $\hat{\mathcal{U}}$. According to Rule 4, the algorithm identifies vertices in $\hat{\mathcal{U}}$ that cannot reach any vertex in $\hat{\mathcal{P}}$, these vertices are classified into $\hat{\mathcal{N}}$ (lines 11-12). Finally, the algorithm calculates the number of vertices which have been classified from $\mathcal{U}$ into either $\mathcal{Y}$ or $\mathcal{N}$. This calculation aids in determining $\mathrm{cnt}_{\mathrm{Y}}$ or $\mathrm{cnt}_{\mathrm{N}}$ for the question (line 13). The algorithm then returns these updated $\mathrm{cnt}(x)$ along with the updated candidates and potential targets (line 14).

Example 5. *Figure 4 and Table 2 show a complete instance of* k-EIS *algorithm for single target in G, where the target $\mathcal{T} = \{v_8\}$. Initially, as shown in Table 2, the vertex with the largest* gain, *v_2, is queried first. Since there is no path from v_2 to v_8, $\mathrm{Q}(v_2) = \text{No}$ is obtained. This leads to updates in the potential targets $\mathcal{P} \leftarrow \mathcal{P} \cap$*

Table 2: Values of $\mathsf{cnt_Y}$, $\mathsf{cnt_N}$, and gain for vertices across questions using k-EIS to find the single target in Figure 4.

Progress		v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
Initial	$\mathsf{cnt_Y}$	1	6	5	7	8	9	9	9	9
Initial	$\mathsf{cnt_N}$	9	4	4	2	2	1	1	1	1
Initial	gain	9	24	20	14	16	9	9	9	9
After $\mathrm{Q}(v_2)$	$\mathsf{cnt_Y}$	-	-	1	3	-	5	5	5	-
After $\mathrm{Q}(v_2)$	$\mathsf{cnt_N}$	-	-	4	2	-	1	1	1	-
After $\mathrm{Q}(v_2)$	gain	-	-	4	6	-	5	5	5	-
After $\mathrm{Q}(v_4)$	$\mathsf{cnt_Y}$	-	-	2	-	-	-	-	2	-
After $\mathrm{Q}(v_4)$	$\mathsf{cnt_N}$	-	-	2	-	-	-	-	2	-
After $\mathrm{Q}(v_4)$	gain	-	-	4	-	-	-	-	4	-

$\{V \setminus \mathsf{des}(v_2)\}$. *According to Rule 2, vertices* $\{v_5, v_9\}$ *are classified into No-candidates. By Rule 3, vertex* $\{v_1\}$ *is classified into Yes-candidates. The Uncertain-candidates* $\mathcal{U}$ *then includes* $\{v_3, v_4, v_6, v_7, v_8\}$. *The next question addresses vertex* v_4 *from* $\mathcal{U}$, *which has the largest* gain *after* $\mathrm{Q}(v_2)$. *Receiving* $\mathrm{Q}(v_4) = \mathsf{Yes}$ *leads to an update in* $\mathcal{P}$ *to* $\mathcal{P} \cap \mathsf{des}(v_4)$, *thus* $\mathcal{P} = \{v_4, v_8\}$. *Applying Rule 4, vertices* $\{v_6, v_7\}$ *are classified into No-candidates. Now,* $\mathcal{U}$ *is reduced to* $\{v_3, v_8\}$, v_3 *and* v_8 *have same* gain, *arbitrarily query* v_3 *or* v_8, *ultimately confirming* v_8 *as the final target.*

Complexity analysis. The SingleEvalSync in Algo. 2 takes $O(nm)$ time and $O(m)$ memory to generate a question. For each vertex v in $\mathcal{U}$, $\mathsf{gain}(v)$ takes $O(m)$ time by performing a multi-source Breadth-First Search. Overall, the k-EIS algorithm in Algo. 1 equipped with Algo. 2 takes $O(tnm)$ time and $O(m)$ memory for finding the single target by asking t questions.

6 MULTIPLE TARGETS SEARCH

In this section, we first propose the k-EIS algorithm to find exact k targets in DAGs. To further improve the effectiveness, we propose a subgraph partition based method Mix-EIS algorithm. The general idea is to separate G into k disjoint subgraphs such that each subgraph has a single target by k-EIS algorithm and apply the single target search algorithms in each subgraph. Finally, we analyze the time complexity of our proposed methods.

6.1 Multiple Target Search Algorithm

We propose k-EIS algorithm for multiple targets search.

k-EIS **algorithm for multiple targets.** The k-EIS algorithm combines the k-EIS framework and algorithm MultipleEvalSync in Algo. 3. Initially, MultipleEvalSync duplicates the sets $\mathcal{P}$, $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$ into temporary sets $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, and $\hat{\mathcal{U}}$ (line 1). If $\mathrm{Q}(x) = \mathsf{Yes}$, it applies Rule 1 to update the sets as follows (lines 2-4): it identifies all ancestors of the queried vertex x in $\hat{\mathcal{U}}$,i.e., $\mathsf{anc}(x) \cap \hat{\mathcal{U}}$, these vertices are classified from $\hat{\mathcal{U}}$ into $\hat{\mathcal{Y}}$ (line 3). Subsequently, the potential targets $\hat{\mathcal{P}}$ is updated to exclude these ancestors except x itself (line 4). If $\mathrm{Q}(x) = \mathsf{No}$, according to Rule 2, the updates proceed differently (lines 5-7): All descendants of x that are still in $\hat{\mathcal{U}}$ are determined,i.e., $\mathsf{des}(x) \cap \hat{\mathcal{U}}$, and classified into $\hat{\mathcal{N}}$, indicating they cannot reach any target (line 6). The $\hat{\mathcal{P}}$ is also refined to exclude all descendants of x (line 7). Finally, the algorithm then calculates the total number of vertices which have been updated from $\hat{\mathcal{U}}$ to either $\hat{\mathcal{Y}}$ or $\hat{\mathcal{N}}$ in this question, which assists in computing the values $\mathsf{cnt_Y}$ or $\mathsf{cnt_N}$ for the question (line 8). The algorithm then returns these updated $\mathsf{cnt}(x)$ along with the updated candidates and potential targets (line 9).

Algorithm 3 MultipleEvalSync

Input: Queried vertex x with $\mathrm{Q}(x)$, $\mathcal{P}$, $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$.
Output: $\mathsf{cnt}(x)$ for the question, $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, $\hat{\mathcal{U}}$

1: $\hat{\mathcal{P}} \leftarrow \mathcal{P}$, $\hat{\mathcal{Y}} \leftarrow \mathcal{Y}$, $\hat{\mathcal{N}} \leftarrow \mathcal{N}$, $\hat{\mathcal{U}} \leftarrow \mathcal{U}$;
2: **if** $\mathrm{Q}(x) = \mathsf{Yes}$ **then**
3: $\quad \mathcal{R} \leftarrow \mathsf{anc}(x) \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{Y}} \leftarrow \hat{\mathcal{Y}} \cup \mathcal{R}$; ▷ By Rule 1
4: $\quad \hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cap \{V \setminus \{\mathsf{anc}(x) \setminus x\}\}$;
5: **else**
6: $\quad \mathcal{R} \leftarrow \mathsf{des}(x) \cap \hat{\mathcal{U}}$, $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{R}$, $\hat{\mathcal{N}} \leftarrow \hat{\mathcal{N}} \cup \mathcal{R}$; ▷ By Rule 2
7: $\quad \hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cap \{V \setminus \mathsf{des}(x)\}$;
8: $\mathsf{cnt}(x) \leftarrow |\mathcal{U}| - |\hat{\mathcal{U}}|$;
9: **return** $\mathsf{cnt}(x)$, $\hat{\mathcal{P}}$, $\hat{\mathcal{Y}}$, $\hat{\mathcal{N}}$, $\hat{\mathcal{U}}$;

Complexity analysis. The k-EIS algorithm in Algo. 1 equipped with Algo. 3 takes $O(tnm)$ time with $O(m)$ memory to find multiple targets by asking t questions. We can optimize to pre-store the reachability of each vertex with $O(n^2)$ memory, which reduces the time complexity from $O(tnm)$ to $O(tn^2)$. Thus, k-EIS can take $O(tn^2)$ time in $O(n^2)$ memory.

6.2 An Improved Algorithm for Multiple Target Search using Divide-and-Conquer

The k-EIS algorithm for multiple targets is designed to address k-EIS problem but requires lots of questions due to the low-valuable gain of Yes answer. When we ask a question on vertex x, if $\mathrm{Q}(x) = \mathsf{No}$, k-EIS algorithm effectively excludes all descendants of the queried vertex both in the single and multiple targets scenarios. However, in the scenario of multiple targets , if $\mathrm{Q}(x) = \mathsf{Yes}$, while confirming the reachability from the queried vertex to at least one target, it affects only the ancestors of the queried vertex, providing limited gain. Therefore, in the scenario of multiple targets, the information gained from each question is significantly less compared to the scenario of single target. To tackle it, the key idea is to divide the DAG into k disjoint subgraphs, provided it is guaranteed that *exactly one target* exists within each subgraph. Then, it is allowed to independently apply single target search algorithms such as our k-EIS, IGS, and TS-IGS to each subgraph. Figure 5 shows an example of our key idea. As some questions asked, we have the candidates $\mathcal{Y}$, $\mathcal{N}$, $\mathcal{U}$ and potential targets $\mathcal{P}$. Then, based on potential targets, our algorithm identifies that G can be separated into three disjoint subgraphs $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$. Finally, we could independently apply the single target search to find targets t_1, t_2, t_3 effectively in each subgraph. Next, we discuss how to divide the original graph into k disjoint subgraphs such that each subgraph has a single target. As the questions asked, potential targets are gradually pruned and the graph can be separated into smaller components using a union-find data structure to efficiently identify and group connected vertices. The detail of subgraphs division algorithm is shown in Algo. 4.

Subgraphs division. Initialize a list of subgraphs, each represented as $\mathcal{G}_i = (V_i, E_i)$. Each subgraph starts with a single vertex from the potential targets $\mathcal{P}$ (line 2). Then iterate through each vertex v in $\mathcal{P}$ and examine each vertex u of its out-neighbors $\mathsf{N^+}(v)$. If an out-neighbor u also belongs to $\mathcal{P}$ and is not already in the same subgraph as v, merge the subgraphs containing v and u. This

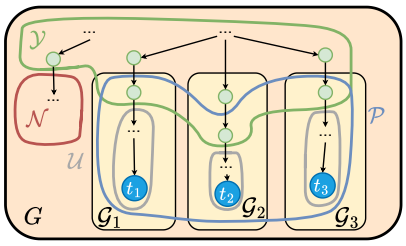


Figure 5: An example of Find-kSingle-DAGs algorithm on a DAG G. The G has the hidden targets $\mathcal{T} = \{t_1, t_2, t_3\}$, after asking some questions, G can be separated into three subgraphs $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$ such that each subgraph has a target.

Algorithm 4 Find-kSingle-DAGs

Input: $G = (V, E)$, potential targets $\mathcal{P}$.
Output: List of subgraphs $\{\mathcal{G}_1, ..., \mathcal{G}_c\}$.

1: Initialize a list of subgraphs, where each $\mathcal{G}_i = (V_i, E_i)$;
2: Initialize each V_i with a single vertex $v \in \mathcal{P}$;
3: **for** each vertex $v \in \mathcal{P}$ **do**
4: **for** each vertex $u \in \mathrm{N}^+(v)$ **do**
5: **if** $u \in \mathcal{P}$ and u not in the same V_i as v **then**
6: Merge V_i and V_j containing v and u, respectively;
7: **for** each subgraph $\mathcal{G}_i = (V_i, E_i)$ **do**
8: Update $E_i = \{\langle v, u\rangle \mid v, u \in V_i, \langle v, u\rangle \in E\}$;
9: **return** $\{\mathcal{G}_1, \ldots, \mathcal{G}_c\}$;

merging step uses a union-find data structure to efficiently manage the combination of subgraphs, ensuring that all connected potential targets are merged together (lines 3-6). For each subgraph $\mathcal{G}_i$, define its set of edges E_i to include only those edges that connect vertices within the same subgraph V_i. This step preserves the connectivity of the original graph within each identified subgraph (lines 7-8). Return the list of disjoint subgraphs $\{G_1, \ldots, G_c\}$, each subgraph contains targets (line 9).

Mix-EIS **algorithm**. We integrate all techniques in Algo. 1, 2, 3, 4 and propose Mix-EIS algorithm. The detail of Mix-EIS algorithm is shown in Algo. 5. Initially, the algorithm establishes four different subsets from V, and an integer c represents the number of subgraphs such that each has at least a target (line 1). Before asking any question, G is the only graph that has targets, so $c = 1$. Based on k-EIS framework, the algorithm calculates gain of each vertex v in Uncertain-candidates, then selects the v^* which has the largest gain to ask a question and updates the potential targets and three candidate sets $\{\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}\}$ by invoking MultipleEvalSync in Algo. 3 (lines 3-5). After updating the information of the question asked, the algorithm identifies disjoint subgraphs with targets in each subgraph and updates c by using the Find-kSingle-DAGs algorithm in Algo. 4 (line 6). If the number of disjoint subgraphs equals k, it determines each subgraph has a single target; the loop will end. (line 2) For each subgraph $\mathcal{G}_i$, the algorithm will independently find a single target t_i by using k-EIS algorithm for a single target by Algo. 1 and Algo. 2 (line 8). Finally, the set of targets $\mathcal{T}^*$ as the output of the Mix-EIS algorithm (line 10).

Algorithm 5 Mix-EIS

Input: $G = (V, E)$, an integer k.
Output: Targets $\mathcal{T}$ with $|\mathcal{T}| = k$.

1: Let $c = 1$, $\mathcal{Y} \leftarrow \{r\}$, $\mathcal{N} \leftarrow \emptyset$,$\mathcal{U} \leftarrow \{V\} \setminus \{r\}$, $\mathcal{P} \leftarrow \{V\} \setminus \{r\}$;
2: **while** the number of disjoint subgraphs $c \neq k$ **do**
3: Calculate gain(v),$v \in \mathcal{U}$, using MultipleEvalSync in Algo. 3;
4: Select $v^* \leftarrow \arg\max_{v \in \mathcal{U}} \text{gain}(v)$ and ask the question $\mathrm{Q}(v^*)$;
5: Update $\{\mathcal{P}, \mathcal{Y}, \mathcal{N}, \mathcal{U}\}$ by invoking Algo. 3;
6: Identify subgraphs $\{\mathcal{G}_1, ..., \mathcal{G}_c\}$ and update c by invoking Find-kSingle-DAGs $(G, \mathcal{P})$ in Algo. 4;
7: **for** $\mathcal{G}_i$ **do**
8: Search each single target $t_i \in \mathcal{G}_i$ by invoking k-EIS in Algo. 1 equipped with Algo. 2;
9: $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{t_i\}$;
10: **return** $\mathcal{T}^*$;

6.3 Complexity Analysis

In this section, we give a detailed complexity analysis of Mix-EIS.

We analyze the Mix-EIS algorithm in two phases for asking $\hat{t} \in \mathbb{Z}^+$ questions. First, Mix-EIS asks $t_1 \in \mathbb{Z}^+$ questions to identify k subgraphs. Then, it further uses $t_2 \in \mathbb{Z}^+$ additional questions to find the exact k targets in each identified subgraph. Here, $\hat{t} = t_1 + t_2$. During the first t_1 questions, each question utilizes k-EIS for multiple targets and the Find-kSingle-DAGs algorithm. The Find-kSingle-DAGs algorithm, outlined in Algo. 4, requires $O(m)$ time to merge connected potential targets into components by examining the neighbors of each vertex in $\mathcal{P}$. After asking t_1 questions, the search in each subgraph can be considered as the single target problem, thus using k-EIS for search single target. Overall, the Mix-EIS combines $O(t_1 nm)$ from k-EIS for multiple targets, $O(t_1 m)$ from Find-kSingle-DAGs algorithm and $O(t_2 nm)$ from k-EIS for single target search. Therefore, the Mix-EIS in Algo. 5 requires $O(t_1 nm + t_1 m + t_2 nm) \subseteq O(\hat{t} nm)$ time with $O(m)$ memory by asking $\hat{t}$ questions where $\hat{t} = t_1 + t_2$. In practice, the number of $\hat{t}$ questions asked by Mix-EIS is usually much less than the number of t questions asked by k-EIS, i.e., $\hat{t} \ll t$, as demonstrated in Figure 9 in Section 8.

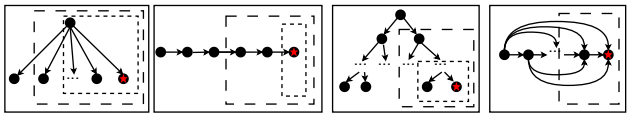
7 COMPARATIVE ANALYSIS IN THEORY

In this section, we compare our method with existing interactive graph search algorithms on various shapes of DAGs in Figure 6, including the DFS-tree-based methods IGS [31], TS-IGS [35], and BinG [4, 19]. Note that we do not include a comparison with the approximate method kBM-IGS [36] here. In addition, BinG only works on tree-structured data.

The worst cases. In the arbitrary graph for searching multiple targets, the worst case is that all vertices are feasible targets. Thus, any algorithm needs to ask questions on all vertices in the worst case as proven in [36]. As the worst case in multiple targets scenario is $O(n)$ for any graph, we consider the single target search problem only in the following.

Star, path, and perfect binary tree. We consider three tree-structure of DAGs: *star*, *path*, and *perfect binary tree*. In the scenario of single target, when the graph G is a star, a central root vertex connects directly to all other leaf vertices, requiring to ask individual questions for each leaf vertex to ensure the target is found,

Shapes of DAG	BinG [4, 19]	IGS/TS-IGS [31, 35]	Ours
star graph	$O(n)$	$O(n)$	$O(n)$
path graph	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
perfect binary tree	$O(\log_2 n)$	$O(\log_2 h(1+\log_2 n)+\log_2 n)$	$O(\log_2 n)$
tournament DAG	N.A.	$O(\log_2 n)$	$O(\log_2 n)$
bipartite graph in Fig. 7	N.A.	$O(n)$	$O(\log_2 n)$



(a) star graph (b) path graph (c) perfect binary tree (d) tournament DAG

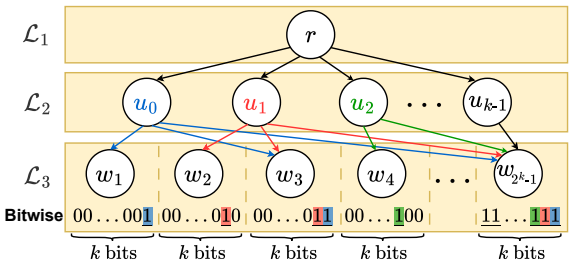
Figure 6: Expected number of needed questions to find single target on the different reasonable shapes of DAG, where n represents the number of vertices, h represents the longest path from root to leaf. The dashed box indicates the reduction in potential targets following the first and second questions.

resulting in $O(n)$ questions. Next, we consider a DAG of path. Each vertex, except for the root and the leaf, has exactly one in-neighbor and one out-neighbor, forming a simple linear path. Algorithms based on binary search can efficiently identify a single target with $O(\log_2 n)$ questions. For perfect binary trees, previous work [31] has demonstrated that any algorithm requires at least $O(\log_2 n)$ questions, with IGS needing at most $O(\log_2 h(1+\log_2 n)+\log_2 n)$ questions. In contrast, our method requires to ask only $O(\log_2 n)$ questions.

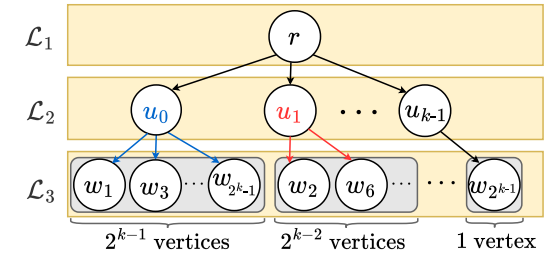
Dense DAGs. The above three tree shapes can be considered as sparse DAGs, which motivates us to consider the algorithm performance on dense DAGs. For dense DAGs, we analyze the tournament DAG and a specific case of bipartite graph. A tournament DAG is defined as $G=(V,E)$, for every pair of vertices v_i and v_j with the ID order $i<j$, there exists a directed edge from vertex i to j, i.e. $E=\{\langle v_i,v_j\rangle \mid v_i,v_j\in V, i<j\}$. In the tournament DAG, DFS-tree-based methods such as IGS and TS-IGS extract a path graph, requiring $O(\log_2 n)$ questions. Our Mix-EIS method can always find the queried vertex, effectively halving the potential targets, thus also requiring $O(\log_2 n)$ questions.

Next, we consider a specific case of bipartite graphs illustrated in Figure 7(a). For this complex dense DAG, Mix-EIS *takes only* $O(\log_2 n)$ *questions*, whereas *DFS-tree-based methods take* $O(n)$ *questions*. Let's construct a structured DAG $G=(V,E)$ with three layers $\mathcal{L}_1,\mathcal{L}_2,\mathcal{L}_3$, as depicted in Figure 7(a). $\mathcal{L}_1$ consists of a root vertex, denoted by $V_1=\{r\}$. $\mathcal{L}_2$ contains k vertices represent the k bits, denoted by $V_2=\{u_0,u_1\cdots,u_{k-1}\}$. $\mathcal{L}_3$ comprises 2^k-1 vertices of bitwise integer, denoted by $V_3=\{w_1,w_2\cdots,w_{2^k-1}\}$. Thus, $V=V_1\cup V_2\cup V_3$. For the edges of G, r connects all the vertices in V_2, i.e., $\mathsf{N}^+(r)=V_2$, and each u_i in $\mathcal{L}_2$ connects to the w_j in $\mathcal{L}_3$ that i-th bit of j is 1, i.e., $\mathsf{N}^+(u_i)=\{w_j \mid j \bmod 2^{i+1}\geq 2^i, 1\leq j\leq 2^k-1\}$. For example, w_5 is connected to u_0 and u_2. In summary, G has a total of $O(2^k)$ vertices and $O(k2^k)$ edges. For the DFS-tree-based methods, here is a DFS-tree extracted from G, as shown in Figure 7(b), in the special case, present vertices with out-degrees as high as $O(2^{k-1})$. Next, we prove in the DAG G, DFS-tree-based methods ask $O(2^{k-1})$ questions, while our method asks $O(k)$ questions.

Lemma 7.1. *In any DFS-tree extracted from the DAG in Figure 7(a), there exists a vertex with* 2^{k-1} *children.*



(a) An example of dense DAG with three layers.



(b) DFS-tree derived from the DAG.

Figure 7: An example of dense DAG and its DFS-tree used in IGS. In the DAG, each u_i in $\mathcal{L}_2$ connects to the w_j in $\mathcal{L}_3$ that i-th bit of j is 1, i.e., $\mathsf{N}^+(u_i)=\{w_j \mid j \bmod 2^{i+1}\geq 2^i, 1\leq j\leq 2^k-1\}$. For example, w_5 is connected to u_0 and u_2.

Proof. Each vertex $v\in\mathcal{L}_2$ has $|\mathsf{N}^+(v)|=2^{k-1}$. The first vertex accessed in $\mathcal{L}_2$ during any DFS traversal will have 2^{k-1} children, essentially forming a star graph. □

Lemma 7.2. *For the DAG in Figure 7(a), any DFS-tree-based method requires* $O(2^{k-1})$ *questions.*

Proof. By Lemma 7.1, the subgraph formed by the first vertex in $\mathcal{L}_2$ and its children can be viewed as a star graph, necessitating $O(2^{k-1})$ questions in any DFS-tree. □

Lemma 7.3. *For the DAG in Figure 7(a), our method requires* $O(k)$ *questions to find the target.*

Proof. Consider the Uncertain-candidates ($\mathcal{U}$) split into two candidates: Uncertain-candidates in $\mathcal{L}_2$ denoted by $\mathcal{U}_2$, Uncertain-candidates in $\mathcal{L}_3$ denoted by $\mathcal{U}_3$, and $\mathcal{U}=\mathcal{U}_2\cup\mathcal{U}_3$. Before asking any questions, $\mathcal{U}_2=V_2$ has k vertices, $\mathcal{U}_3=V_3$ has 2^k-1 vertices. If we keep asking questions on the vertices in $\mathcal{U}_2$, irrespective of whether the answers are Yes or No, each question has two effects: (1) it directly reduces the current size of $\mathcal{U}_2$ by one, due to the queried vertex itself. (2) it reduces $\mathcal{U}_3$ to half its original size, i.e. $|\mathcal{U}_3|\leftarrow\frac{|\mathcal{U}_3|}{2}$ due to each vertex in $\mathcal{U}_2$ connects half of vertices in $\mathcal{U}_3$. Consequently, after asking k consecutive questions on the vertices in $\mathcal{U}_2$, $\mathcal{U}_2$ will be an empty set and $\mathcal{U}_3$ will be an empty set or last one vertex. If $\mathcal{U}_3$ is an empty set, obviously we can find the target. If $\mathcal{U}_3$ last one vertex w_{last}, we need to ask one more question on w_{last} to determine the target is either w_{last} itself or w_{last}'s parents in $\mathcal{L}_2$. In summary, it requires k or $k+1$ questions to find any target in Figure 7 by querying k vertices in $\mathcal{L}_2$. Our method will ask questions as the above questions' sequence and asks $O(k)$ questions to find the target in G. □

Table 3: Summary of datasets.

Dataset	Vertices ($\|V\|$)	Edges ($\|E\|$)	Max Degree	Data Domain
ProdClass	3,616	3,615	95	Tree
Amazon	29,240	29,239	225	Tree
Yago	493,839	493,838	44,538	Tree
ACM_CCS	1,928	2,113	18	DAG
Wiki_Edits	1,587	5,343	1,418	DAG
ImageNet	74,401	75,850	402	DAG

Theorem 7.4. *For the DAG shown in Figure 7(a), in the single target scenarios, all the DFS-tree-based methods, e.g., IGS and TS-IGS ask $O(n)$ questions and our method asks $O(\log_2 n)$ questions.*

Proof. By Lemmas 7.2 and 7.3, we can prove Theorem 7.4. □

In summary, our method is applicable for finding multiple targets on DAGs. In the case of dense DAGs in Figure 7(a), our method based on the newly proposed gain function and Uncertain-candidates outperforms the DFS-tree-based methods in finding the single target in Theorem 7.4. Our method costs $O(\log_2 n)$ questions, but DFS-tree-based methods need $O(n)$ questions, reflecting the significant effectiveness of our method in theory.

8 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our proposed methods. All algorithms are implemented in C++. The experiments are carried out on a server running Oracle Linux 8.8, equipped with a Xeon Gold 6330 processor at 2.0 GHz and 2TB of RAM.

Datasets. We evaluate our algorithms using six real-world datasets, each selected for their relevance to graph algorithms and their distinct structural characteristics.

- **ProdClass**[1]: A product classification standard from the National Bureau of Statistics of China. This dataset categorizes products for statistical purposes across various economic activities.
- **Amazon** [11]: Contains hierarchical product categories from Amazon. This dataset traces paths from the root to specific product categories within Amazon's extensive product catalog.
- **Yago** [1, 24]: Contains taxonomy vertices from Wikipedias.
- **ACM_CCS**[2]: Represents the 2012 ACM Computing Classification System. It is a poly-hierarchical ontology to support semantic web applications and enhance the ACM Digital Library's search capabilities.
- **Wiki_Edits** [16]: Captures the bipartite edit network from the Kabiye Wikipedia. This network links users and pages through edit events, depicting interactions within the platform.
- **ImageNet** [5]: A hierarchical image database based on the WordNet [9]. It features extensive image categories with detailed ground truth labels for image classification tasks.

For clarity and to avoid redundancy, the specific details about vertices, edges, max degree and structures of these datasets are summarized in Table 3.

[1]https://www.stats.gov.cn/sj/tjbz/tjypflml/2010/
[2]https://dl.acm.org/ccs

Table 4: The average number of questions for finding a single target in k-EIS problem.

Dataset	IGS	TS-IGS	BinG	KBM-IGS	1-EIS
ProdClass	34.80	34.22	30.93	30.93	**30.93**
Amazon	30.18	29.15	24.97	24.98	**24.97**
Yago	5171.46	5170.94	5167.73	5168.23	**5167.74**
ACM_CCS	18.65	17.96	×	×	**14.22**
Wiki_Edits	639.99	639.01	×	×	**39.96**
ImageNet	38.60	37.13	×	×	**31.43**

Comparison methods. We compare four state-of-the-art competitors for interactive graph search as follows.

- **IGS** [31]: This method identifies a single target in a DAG by extracting a DFS-tree from the DAG and split the tree into several connected paths.
- **TS-IGS** [35]: Building on IGS, TS-IGS also utilizes heavy-path decomposition but introduces a result-sensitive binary search technique.
- **BinG** [4, 19]: Based on a greedy strategy, this method locates a single target in a tree by querying vertices that most significantly reduce the graph's size.
- **KBM-IGS** [36]: Aimed at finding multiple targets in a tree, it uses dynamic programming to select the best vertex with the highest expected gain, providing a balanced trade-off between target probability and benefit gain.

We also evaluate three of our developed algorithms:

- 1-EIS: The k-EIS method for single target search in Algorithm 1 equipped with Algorithm 2.
- k-EIS: The k-EIS method for multiple targets search in Algorithm 1 equipped with Algorithm 3.
- Mix-EIS: Our subgraph partition based method for multiple targets search in Algorithm 5.

Evaluation metrics and parameter settings. We employ two principal metrics to evaluate our algorithms:

(1) **The number of questions**: This metric counts the number of questions needed to identify single or multiple targets. In k-EIS problem, the number of questions can be seen as the cost of problem. So, the fewer questions, the less cost.
(2) **The size of potential targets [28, 31, 36]**: This metric reflects the number of vertices that could potentially be targets, which denoted by $|\mathcal{P}|$, evaluated under a scenario with a fixed number of questions. In works [28, 31], it is called candidate set size (CSS for short) in their experiments.

For the scenario of single target, we randomly select 1,000 vertices from the Amazon and ImageNet datasets to serve as targets in 1,000 separate trials. For other datasets, each vertex is considered as a target in separate groups. For the scenario of multiple targets, we set $k = 2, 3, 4, 5$ and conduct 200 test cases for each dataset, with targets randomly generated. **We set $k = 3$ by default.** We denote the result as ×, if the method is inapplicable in the dataset. For all algorithms, we precompute the first z questions offline by taking 2^z states, no matter which the answer of each question is. Thus, it can directly get the next vertex for asking question in the first z rounds, where we set a small integer $z = 5$ by default.

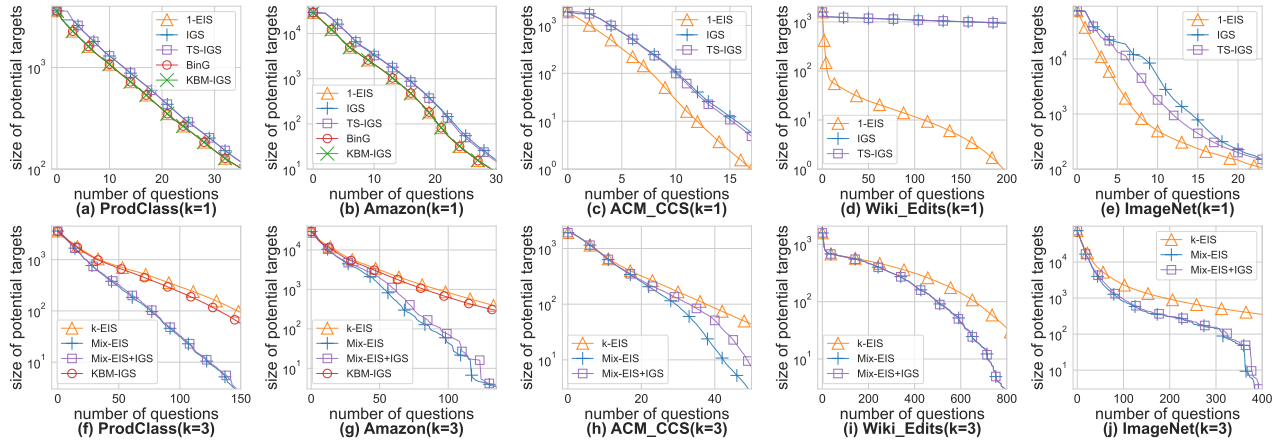


Figure 8: The size of potential targets with the fixed number of questions in the single or three targets scenarios.

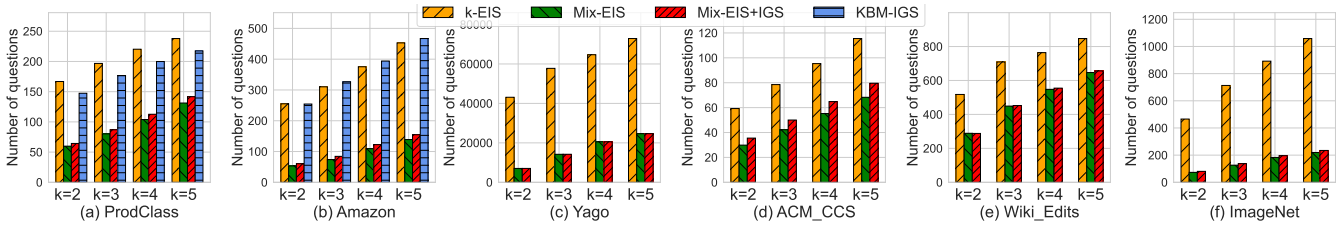


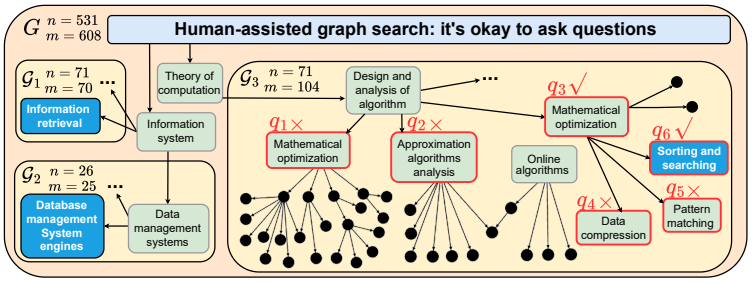
Figure 9: The average number of questions for different algorithms with different k in the k-EIS problem.

EXP-1: Quality evaluation of finding a single target in k-EIS problem. Our 1-EIS algorithm undergoes extensive evaluation against IGS, TS-IGS, BinG, and KBM-IGS, focusing on an average number of questions and the ability to reduce the size of potential targets within diverse datasets. Detailed results are presented in Table 4 and illustrated in Figure 8(a)-(e). In tree-structured datasets, 1-EIS demonstrates effectiveness comparable to that of BinG and KBM-IGS, it achieves a significantly lower average number of questions (24.97 and 30.93, respectively), outperforming IGS and TS-IGS. For DAG datasets, 1-EIS consistently achieves the best performance regarding average questions and size of potential target reduction. The Wiki_Edits dataset, characterizes by its dense DAG structure, 1-EIS not only reduces the average questions to 39.96—a substantial improvement over other methods—but also demonstrates an exceptionally rapid reduction in the size of potential targets.

EXP-2: Quality evaluation of the k-EIS problem. In the k-EIS problem, we propose k-EIS algorithm to find the exact k targets in a DAG and the Mix-EIS algorithm to improve effectiveness. In Section 6.2, recall that the final step of the Mix-EIS algorithm is to find a single target in each subgraph, and we can use IGS to replace the 1-EIS algorithm. For comparison, we use KBM-IGS in tree datasets and combine the Mix-EIS algorithm with IGS, referred to as Mix-EIS +IGS algorithm, on all datasets. Specifically, KBM-IGS can't find exact multiple targets in Yago because of the large scale. As shown in Figure 9, Mix-EIS algorithm and Mix-EIS +IGS algorithm perform much better than k-EIS algorithm and KBM-IGS. Furthermore, Mix-EIS algorithm asks less questions than Mix-EIS +IGS algorithm in all datasets. As shown in Figure 8(f)-(j), with a low number of questions, all the algorithms have similar effectiveness in decreasing the potential targets. With the larger number of questions, the effectiveness has diverged, especially between Mix-EIS algorithm and k-EIS algorithm. Mix-EIS algorithm and Mix-EIS +IGS algorithm have a similar trend in all datasets, specifically, in Amazon and ACM_CCS, Mix-EIS algorithm has significantly better performance.

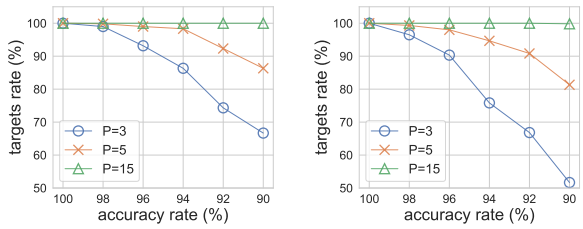
EXP-3: Case study of finding multiple targets on ACM_CCS. We conduct a case study on the ACM_CCS to identify specific index terms related to the article titled "Human-assisted graph search: it's okay to ask questions" [3]. The article targets terms such as "Information retrieval", "Database management System engines", and "Sorting and searching" ($k = 3$). We extracted a sub-DAG G from the dataset from "information systems" and "Theory of computation", consisting of 531 vertices and 608 edges. The extracted graph G is visualized in Figure 10. Through Mix-EIS algorithm's initial ten questions, it identified three disjoint subgraphs containing the targets, which we denote as $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$. $\mathcal{G}_1$ is a tree with 26 vertices and 25 edges; $\mathcal{G}_2$ is another tree with 71 vertices and 70 edges; and $\mathcal{G}_3$ is a more complex DAG with 71 vertices and 104 edges, rooted in "design and analysis of algorithm". In detail, we discuss finding the single target in the most complex subgraphs $\mathcal{G}_3$. The sequence of questions and their corresponding answers in this subgraph are detailed in the table within Figure 10. It only costs six questions to find the target "Sorting and searching" in $\mathcal{G}_3$. On the contrast, IGS needs ten questions to find the same target in $\mathcal{G}_3$. Totally, to find the targets in G, k-EIS algorithm costs

[3]https://dl.acm.org/doi/10.14778/1952376.1952377



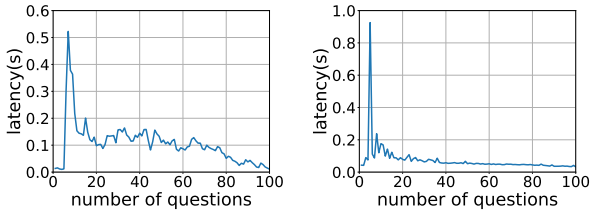
Q	Q	cnt_Y	cnt_N	$\lvert\mathcal{U}\rvert$
Initial	/	/	/	71
q_1	No	48	23	47
q_2	No	38	8	39
q_3	Yes	34	6	5
q_4	No	5	1	4
q_5	No	4	1	3
q_6	Yes	3	1	0

Figure 10: Case study on ACM_CCS. Here, $k = 3$, and the targets are $\mathcal{T}$ = {"Information retrieval", "Database management System engines", "Sorting and searching"}. After asking ten questions in G, G separates into three disjoint subgraphs $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$. In the $\mathcal{G}_3$, Mix-EIS algorithm only costs six questions to find the exact target "Sorting and searching".



(a) ACM_CCS (b) Amazon

Figure 11: Quality evaluation with wrong answers.



(a) ImageNet (b) Yago

Figure 12: The latency of the Mix-EIS algorithm to determine the next question on ImageNet and Yago.

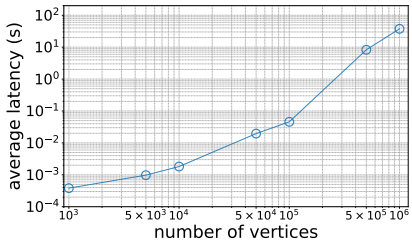


Figure 13: Efficiency evaluation of Mix-EIS on the DAGs.

59 questions, Mix-EIS +IGS algorithm costs 43 questions, and our Mix-EIS algorithm only costs 36 questions, which performs best in the case study. It displays that our Mix-EIS algorithm has perfect effectiveness int real-world applications.

EXP-4: Quality evaluation of handling wrong answers. We conduct a quality evaluation of Mix-EIS to handle wrong answers. To reduce the error, we use the strategy of majority voting to ask $P \in \mathbb{Z}$ users on the same question and then take the majority voting answer as the final answer. In this experiment, we vary the *accuracy rate* from 100% to 90%, which is the probability of answering wrongly by a user. We test three cases for $P = 3, 5, 15$. Figure 11 shows that Mix-EIS finds more than 80% of targets and all targets for $P = 3$ and $P = 15$, respectively, even for the accuracy of correct answer is 90%.

EXP-5: Latency evaluation of interactive search. We evaluate the latency of Mix-EIS in determining the next question on ImageNet and Yago. Similar results on other datasets can be observed. As shown in Figure 12, Mix-EIS takes the maximum latency of less than one second at the sixth question on both datasets. Note that we have precomputed the first five questions offline for all competitive methods by default. Moreover, with more questions asked, the decision latency of Mix-EIS generally decreases. The average latency of Mix-EIS takes 0.129 seconds on ImageNet and 0.015 seconds on Yago, respectively. The fast latency results demonstrate the feasibility of our interactive search for multiple targets, which can provide a user-friendly service.

EXP-6: Scalability test.

We conducted a scalability test for Mix-EIS on the increasing-sized DAGs. We randomly generated seven DAGs by varying the vertex size from 10^3 to 10^6 following a power-law distribution, maintaining an edge-to-vertex ratio of 1.02 (similar to the ImageNet). We repeat 100 random cases to search three targets on each graph and report the average time. Figure 13 shows that the average latency increases stable from small DAGs to million-scale DAGs and takes a few seconds on the largest DAG, demonstrating a good scalability performance of Mix-EIS.

9 CONCLUSION

This paper studies the problem of interactive graph search for multiple targets on DAGs. We tackle the problem by proposing a novel k-EIS framework based on uncertain candidates. We design a gain function to select the best vertex for asking questions. Effective rules are derived to update three kinds of candidates and potential targets. Built upon k-EIS framework, we propose an effective method for single target search, which is much better than state-of-the-art methods in worst DAG cases. In addition, we develop Mix-EIS for multiple targets search that can partition the whole DAG into disjoint subgraphs for single target search. Extensive experiments validate the superiority of our k-EIS framework and algorithms against state-of-the-art competitors.

REFERENCES

[1] [n. d.]. https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago.

[2] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, and Mukesh Mohania. 2008. Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 13–22.

[3] Qianhao Cong, Jing Tang, Kai Han, Yuming Huang, Lei Chen, and Yeow Meng Chee. 2022. Noisy Interactive Graph Search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 231–240.

[4] Qianhao Cong, Jing Tang, Yuming Huang, Lei Chen, and Yeow Meng Chee. 2022. Cost-effective algorithms for average-case interactive graph search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1152–1165.

[5] 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*. Ieee, 248–255.

[6] Eyal Dushkin and Tova Milo. 2018. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data*. 1007–1019.

[7] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. 2015. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1015–1030.

[8] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.

[9] Christiane Fellbaum. 1998. *WordNet: An electronic lexical database*. MIT press.

[10] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. 2012. So who won? Dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 385–396.

[11] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[12] Chien-Ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. 2013. Adaptive task assignment for crowdsourced classification. In *International conference on machine learning*. PMLR, 534–542.

[13] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2017. Community search over big graphs: Models, algorithms, and opportunities. In *Proceedings of the 33rd IEEE International Conference on Data Engineering (ICDE)*. 1451–1454.

[14] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers.

[15] Rajesh Jayaram. 2023. Technical Perspective: Optimal Algorithms for Multiway Search on Partial Orders. *ACM SIGMOD Record* 52, 1 (2023), 83–83.

[16] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. 1343–1350.

[17] Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-action-reflection: Towards deep interaction between conversational and recommender systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 304–312.

[18] Yingxue Li, Shaohan Chen, and Weiguo Zheng. 2023. Knowledge-Driven Interactive Graph Search. *Beijing Da Xue Xue Bao* 59, 5 (2023), 735–746.

[19] Yuanbing Li, Xian Wu, Yifei Jin, Jian Li, and Guoliang Li. 2020. Efficient algorithms for crowd-aided categorization. *PVLDB* 13, 8 (2020), 1221–1233.

[20] Yuanbing Li, Xian Wu, Yifei Jin, Jian Li, Guoliang Li, and Jianhua Feng. 2022. Adapative algorithms for crowd-aided categorization. *The VLDB Journal* (2022), 1–27.

[21] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. 2012. Cdas: a crowdsourcing data analytics system. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1040–1051.

[22] Shangqi Lu, Wim Martens, Matthias Niewerth, and Yufei Tao. 2022. Optimal algorithms for multiway search on partial orders. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 175–187.

[23] Shangqi Lu, Wim Martens, Matthias Niewerth, and Yufei Tao. 2023. An Optimal Algorithm for Partial Order Multiway Search. *ACM SIGMOD Record* 52, 1 (2023), 84–92.

[24] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2015. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*.

[25] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. 2012. Counting with the crowd. *Proceedings of the VLDB Endowment* 6, 2 (2012), 109–120.

[26] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 188–197.

[27] Aditya Parameswaran, Stephen Boyd, Hector Garcia-Molina, Ashish Gupta, Neoklis Polyzotis, and Jennifer Widom. 2014. Optimal crowd-powered rating and filtering algorithms. *Proceedings of the VLDB Endowment* 7, 9 (2014), 685–696.

[28] Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. 2011. Human-Assisted Graph Search: It's Okay to Ask Questions. *PVLDB* 4, 5 (2011), 267–278.

[29] Aditya G Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. 2012. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 361–372.

[30] Lu Shangqi, Wim Martens, Matthias Niewerth, and Yufei Tao. 2023. Partial Order Multiway Search. *ACM Transactions on Database Systems* 48, 4 (2023), 1–31.

[31] Yufei Tao, Yuanbing Li, and Guoliang Li. 2019. Interactive graph search. In *Proceedings of the 2019 International Conference on Management of Data*. 1393–1410.

[32] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. 2017. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1133–1148.

[33] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment* 6, 6 (2013), 349–360.

[34] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier Movellan, and Paul Ruvolo. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. *Advances in neural information processing systems* 22 (2009).

[35] Zhuowei Zhao, Junhao Gan, Jianzhong Qi, and Zhifeng Bao. 2024. Efficient Example-Guided Interactive Graph Search. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 342–354.

[36] Xuliang Zhu, Xin Huang, Byron Choi, Jiaxin Jiang, Zhaonian Zou, and Jianliang Xu. 2021. Budget constrained interactive search for multiple targets. *Proceedings of the VLDB Endowment* 14, 6 (2021), 890–902.