Budget Constrained Interactive Search for Multiple Targets

Xuliang Zhu[1], Xin Huang[1], Byron Choi[1], Jiaxin Jiang[1], Zhaonian Zou[2], Jianliang Xu[1]
[1]Hong Kong Baptist University, Hong Kong, China
[2]Harbin Institute of Technology, Harbin, China
{csxlzhu, xinhuang, bchoi, jxjian, xujl}@comp.hkbu.edu.hk, znzou@hit.edu.cn

ABSTRACT

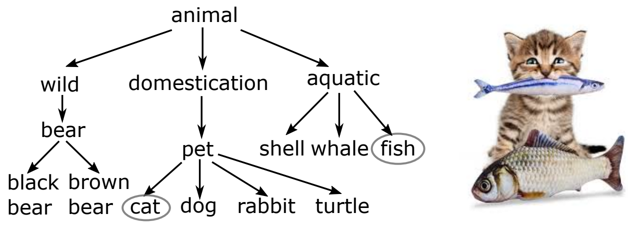
Interactive graph search leverages human intelligence to categorize target labels in a hierarchy, which is useful for image classification, product categorization, and database search. However, many existing interactive graph search studies aim at identifying a single target optimally, and suffer from the limitations of asking too many questions and not being able to handle multiple targets.

To address these two limitations, in this paper, we study a new problem of budget constrained interactive graph search for multiple targets called kBM-IGS problem. Specifically, given a set of multiple targets $\mathcal{T}$ in a hierarchy and two parameters k and b, the goal is to identify a k-sized set of selections $\mathcal{S}$, such that the closeness between selections $\mathcal{S}$ and targets $\mathcal{T}$ is as small as possible, by asking at most a budget of b questions. We theoretically analyze the updating rules and design a penalty function to capture the closeness between selections and targets. To tackle the kBM-IGS problem, we develop a novel framework to ask questions using the best vertex with the largest expected gain, which provides a balanced trade-off between target probability and benefit gain. Based on the kBM-IGS framework, we first propose an efficient algorithm STBIS to handle the SingleTarget problem, which is a special case of kBM-IGS. Then, we propose a dynamic programming based method kBM-DP to tackle the MultipleTargets problem. To further improve efficiency, we propose two heuristic but efficient algorithms, kBM-Topk and kBM-DP+. Experiments on large real-world datasets with ground-truths verify both the effectiveness and efficiency of our algorithms.

PVLDB Reference Format:
Xuliang Zhu, Xin Huang, Byron Choi, Jiaxin Jiang, Zhaonian Zou, Jianliang Xu. Budget Constrained Interactive Search for Multiple Targets. PVLDB, 14(6): 890-902, 2021.
doi:10.14778/3447689.3447694

1 INTRODUCTION

Crowdsourcing, such as Amazon's Mechanical Turk [1] and CrowdFlower [2], allows organizations to design human-aided services in which humans can help solve tasks and get rewards. In real applications, many tasks such as object categorization [23], entity resolution [30, 31], filtering noisy data [14, 25], ranking [21], and labeling [5], are complex and difficult to resolve algorithmically. With regard to human-aided object categorization, the graph search problem concerns leveraging human intelligence to categorize the target labels of a given object in a label hierarchy, which has a wide range of applications including image classification [10, 17], product categorization [22], and relational database search [29].



(a) Hierarchical ImageNet Tree (b) Image

Figure 1: An example of uncategorized image in Figure 1(b) has target labels $\mathcal{T}$ ={"cat", "fish"} in hierarchy in Figure 1(a).

Recently, Tao et al. [29] investigated the problem of *interactive* graph search (IGS) to locate *one unique target vertex* in a hierarchy $\mathcal{H}$, with as few questions as possible. For example, Figure 1(a) shows a hierarchical tree with several labeled vertices. A directed edge from one vertex to another represents the concept-instance relationship, e.g., "pet" is a general concept of four instances "cat", "dog", "rabbit", and "turtle". Note that the target is unknown in advance. To identify the target, interaction is allowed to iteratively ask questions using the vertices in the hierarchy, e.g., "Is this a wild (animal)?", "Is this a pet?". Assuming that the target is "pet" in Figure 1(a), we need to ask at least five questions of "Is this x?", where $x \in$ {"pet", "cat", "dog", "rabbit", "turtle"}, to get the answers {Yes, No, No, No, No} and then determine the exact target of "pet". Effective algorithms with theoretical guarantee are proposed for finding the exact target using at most $\lceil \log_2 h \rceil (1+\lfloor \log_2 n \rfloor)+(d-1)\cdot \lceil \log_d n \rceil$ questions [29], where n, d, h are respectively the number of vertices, the maximum out-degree, and the hierarchy height in $\mathcal{H}$. However, two issues remain open:

- Finding nearly-optimal targets using a constrained budget. IGS [29] may incur a high cost to identify the exact target. It does not limit the number of questions that can be asked. In the worst case, the proposed algorithm asks $(d-1)\cdot \lfloor \log_d n \rfloor$ questions to optimally identify the target. In real hierarchy datasets, the out-degree d could be large, e.g., ImageNet has $d = 391$ and $n = 74,401$ [10]. Thus, users may need to answer 782 questions, which is not very practical. Moreover, asking questions is potentially costly [23], which motivates the problem of budget constrained IGS to bound the total cost.
- Identifying multiple targets. Existing studies on the IGS problem [17, 29] only consider a single target, where the answer has one and only one target. However, in real applications

Proceedings of the VLDB Endowment, Vol. 14, No. 6 ISSN 2150-8097.
doi:10.14778/3447689.3447694

of object categorization, an object may have multiple labels. Even worse, it is difficult to determine in advance how many labels the object may have. For example, Figure 1(b) shows an uncategorized image object. Both "cat" and "fish" are suitable to label the object, but either one alone is not good enough.

To address the above issues, in this paper, we propose a new kBM-IGS problem of interactive graph search for identifying multiple targets $\mathcal{T}$ using a constrained budget to ask at most b questions. Specifically, in each round, our kBM-IGS scheme asks a question in the form "Given a query vertex q in tree $\mathcal{H}$, can vertex q reach one of targets in $\mathcal{T}$?" and receives the answer from human-assisted interactions. On the basis of the previous answers, the kBM-IGS scheme determines the next question to ask. Finally, it selects a set of vertices to represent the targets after b questions are answered.

However, it is significantly challenging to identify the most suitable selections in the kBM-IGS problem, for the following reasons. First, the number of targets is unknown in advance, which may have one or more ground-truth labels. Second, in the worst case, a total of $O(n)$ questions is needed to find the exact targets, which makes the selection of b questions difficult. Third, since the targets are unknown, another challenge is how to measure the goodness of a solution, i.e., the closeness between selections and targets.

In light of the above, our problem is formulated as finding a k-sized selection set of vertices to approach the targets as close as possible, w.r.t. an input number of k and a budget of b questions. For example, consider the hierarchy and the uncategorized image object in Figure 1. Assume that $k = 2$ and $b = 2$. We ask two questions of "Is this x?"[1], where x is "pet" and "fish", respectively, and both answers are Yes. After that, we cannot ask any more questions to verify the other four specified pets, i.e., "cat", "dog", "rabbit", and "turtle". Thus, we select "pet" and "fish" as the solution. Assume that the targets are "cat" and "fish". It can be seen that the selections of "pet" and "fish" are close to the targets, since "pet" is a generalization of "cat". On the other hand, "animal" is also a good label, but it is far from "cat" and worse than our selection "pet".

To tackle the kBM-IGS problem, we propose a novel kBM-IGS framework, which uses a greedy strategy to ask the best question with the largest expected gain at each round. Specifically, vertices have different probabilities to be targets and may get Yes/No answers for questions asked. In general, a vertex at the top level of the hierarchy has a high probability of getting a Yes answer. However, the benefit of getting a Yes answer can be less than a No answer, which implies that none of descendants are targets. Therefore, we propose an expected gain to trade-off the target probability and benefit gain. Thus, the kBM-IGS framework can find the vertex with the largest expected gain to ask the next question. On the basis of the kBM-IGS framework, we first propose an efficient algorithm STIGS to solve the SingleTarget problem, which is a special case of kBM-IGS with $|\mathcal{T}| = 1$. Different from the SingleTarget problem, it is difficult to calculate the gains in the MultipleTargets problem. We then develop a kBM-DP method to calculate the optimal penalty between the k-sized selections and potential targets. To further improve efficiency, we propose two heuristic but efficient algorithms. To summarize, we make the following contributions:

- We propose a new kBM-IGS problem of budget constrained interactive graph search for identifying multiple targets in a hierarchical tree. We raise the problem of finding the k-sized selections close to multiple targets using a constrained number of b questions, and formally design a penalty function to measure the closeness between selections and targets (Section 3).
- We give theoretical analysis of potential targets and Yes candidate, which offers useful updating rules to prune disqualified candidates. On the basis of the updating rules and expected gains, we propose a novel kBM-IGS framework to tackle the kBM-IGS problem by asking b good questions (Section 4).
- We tackle one instance of kBM-IGS problem, the SingleTarget problem, where the target involves a single answer. On the basis of the kBM-IGS framework, we derive new updating rules and propose a greedy algorithm STBIS. (Section 5).
- We propose three efficient algorithms for identifying multiple targets based on the kBM-IGS framework, including a dynamic programming algorithm kBM-DP and two improved fast algorithms kBM-Topk and kBM-DP+ (Section 6).
- We conduct extensive experiments on real-world datasets with *ground-truth multiple targets* to validate the efficiency and effectiveness of proposed framework and algorithms (Section 7).

2 RELATED WORK

Our work is related to human-assisted data processing tasks [13, 16, 24, 25, 30, 32, 33], hierarchy construction [6, 8, 27], and object categorization problems [7, 11, 12, 15, 19, 28, 34]. Table 1 shows a detailed comparison of the three most relevant studies, IGS [29], BinG [17], HGS [23], and our kBM-IGS. Tao et al. [29] propose an *interactive graph search* (IGS) method for identifying a single target in a directed hierarchy. The general idea is to apply heavy-path decomposition to produce a balance representation of hierarchy and tackle the problem by binary searches. Li et al. [17] model the single target problem as a decision tree construction problem. They propose a greedy based method for interactive graph search (denoted as BinG), which improves the performance of IGS [29]. Both studies consider a single target and find the exact result using an unlimited budget of questions. Consider the example shown in Figure 2. Assume that the target is r. Both IGS [29] and BinG [17] would ask all its children to determine whether r is the target, which takes $n - 1$ questions. Different from these two studies [17, 29], our proposed framework can tackle both SingleTarget and MultipleTargets problems and select the representative targets within a bounded budget.

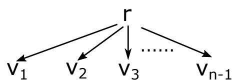


Figure 2: A hierarchy has n vertices with the target r.

Parameswaran et al. [23] also investigate both SingleTarget and MultipleTargets problems with bounded budgets and propose HGS to find multiple targets using b questions. However, the novelty of our problem is the consideration of the *interactive* setting, which enables *dynamic* algorithm designs and brings *significant* performance benefits. First, the HGS scheme is a *non-interactive* algorithm

[1] The question is equivalent to a search question in the form "Given a query vertex q in tree $\mathcal{H}$, can vertex q reach one of targets in $\mathcal{T}$?"

Table 1: Comparison with relevant studies IGS [29], BinG [17], and HGS [23]. Here, b is the budget of questions, and n, d, h are respectively the number of vertices, the maximum out-degree, and the height in the hierarchy.

Method	Interactive	Targets	Budget	Questions (Worst Case)	Time (Each Question)	Time (Total)
IGS [29]	✓	Single	×	$\lceil \log_2 h \rceil (1 + \lfloor \log_2 n \rfloor) + (d-1) \cdot \lceil \log_d n \rceil$	$O(1)$	$O(n \log n)$
BinG [17]	✓	Single	×	$n-1$	$O(n)$	$O(n^2)$
HGS [23]	×	Single	✓	b	/	$O(n \log n)$
	×	Multiple	✓	b	/	$O(b^2 n^6)$
kBM-IGS	✓	Single	✓	b	$O(n)$	$O(bn)$
	✓	Multiple	✓	b	$O(nh^2dk^2)$	$O(bnh^2dk^2)$

that asks all b questions in *one go* and then, based on the workers' answers, does the best to figure out where the targets are. Its objective is to choose the b questions wisely to minimize the size of the candidate set. As a result, it may not be able to find the candidates close to the targets. In contrast, our proposed approach leverages the answers of the previous l questions ($1 \leq l \leq b-1$) to *dynamically* determine the $(l+1)$-th question. Such interaction allows our algorithm to quickly narrow down the search space and efficiently guide the search towards the targets. Second, the HGS algorithm divides the whole hierarchy into b subtrees and asks a question on each root of the b subtrees. It has an extreme time complexity of $O(b^2n^6)$ [23]. Different from HGS, our dynamic approach works by asking one question each time on a single vertex that achieves the largest expected gain based on the previous answers. In other words, our approach is a greedy algorithm that runs b times to identify b questions in $O(bnh^2dk^2)$ time. It is more effective and efficient than HGS, as will be validated by the experiments in Section 7.

3 PRELIMINARIES

In this section, we present definitions and formulate our problem.

3.1 Hierarchical Tree

Let $\mathcal{H} = (V, E)$ be a directed hierarchical tree rooted at r with a set V of vertices and a set E of directed edges, where the root $r \in V$ and the edge set $E = \{\langle v, u \rangle : v \text{ is the parent of } u\}$. Let the height of $\mathcal{H}$ be h and $n = |V|$. For $v \in V$, we denote its children of v by $\mathsf{child}(v) = \{u : \langle v, u \rangle \in E\}$ and its unique parent by $\mathsf{par}(v)$ where $\langle \mathsf{par}(v), v \rangle \in E$. Given two vertices u and v, we say that u can reach v (denoted as $u \longrightarrow v$), if and only if there exists a directed path from u to v in $\mathcal{H}$. If u cannot reach v, we use $u \not\rightarrow v$ to represent it. Note that $v \longrightarrow v$ and $r \longrightarrow v$ for any vertex $v \in V$. Moreover, the distance from u to v is denoted by $\mathsf{dist}\langle u, v \rangle$, as the length of the shortest path from u to v in $\mathcal{H}$. Note that $\mathsf{dist}\langle v, v \rangle = 0$ and $\mathsf{dist}\langle u, v \rangle = +\infty$ if $u \not\rightarrow v$. In addition, the ancestors and descendants of a vertex v are denoted by $\mathsf{anc}(v) = \{u \in V : u \longrightarrow v\}$ and $\mathsf{des}(v) = \{u \in V : v \longrightarrow u\}$, respectively.

Example 1. *Figure 3 shows an example of hierarchical tree H rooted by* v_0. *The children of* v_3 *are* $\mathsf{child}(v_3) = \{v_6, v_7, v_8\}$ *and its parent is* v_1. *We have* $\mathsf{anc}(v_3) = \{v_0, v_1, v_3\}$ *and* $\mathsf{des}(v_3) = \{v_3, v_6, v_7, v_8\}$. *The distance* $\mathsf{dist}\langle v_0, v_3 \rangle = 2$ *and* $\mathsf{dist}\langle v_2, v_3 \rangle = +\infty$.

3.2 kBM-IGS Interactive Scheme

In the following, we introduce the scheme of budget-based interactive search for identifying multiple targets. In contrast to IGS [29] and BinG [17], our kBM-IGS has new rules and features for asking a limited number of questions. The interactive scheme has four components: *targets*, *questions*, Yes-candidates, and *selections*.

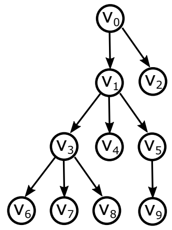


Figure 3: The hierarchical tree used in the running example.

Targets. The targets are a set of vertices in tree $\mathcal{H}$, denoted as $\mathcal{T} \subseteq V$. The goal of our kBM-IGS if finding the target set $\mathcal{T}$, which needs to be identified through a few rounds of question-asking.The targets $\mathcal{T}$ can be chosen arbitrarily from V, which have two characteristics: *variant cardinality* and *target independence*. First, in terms of target cardinality, we categorize $\mathcal{T}$ into two types, SingleTarget and MultipleTargets, following [23]. If the size of $\mathcal{T}$ is known and $|\mathcal{T}| = 1$, we call it SingleTarget. If the size of $\mathcal{T}$ is unknown and variant with $|\mathcal{T}| \geq 1$, we call it MultipleTargets, which does not constrain the size of $\mathcal{T}$. On the other hand, the target set $\mathcal{T}$ must satisfy the property of target independence, that is, any two vertices in $\mathcal{T}$ are not related [23]:

$$\forall v, u \in \mathcal{T}, \text{ if } v \neq u, \text{ then } v \not\rightarrow u.$$

Consider the example shown in Figure 1(a), where we assume the target set is $\mathcal{T}$ ={"cat", "fish"}. Although "pet" is also a correct terminology to represent the object in Figure 1(b), it is not suitable to be added into $\mathcal{T}$, as it would violate the property of target independence as "pet" reaches "cat" in Figure 1(a). Actually, "cat" is a more precise label to describe the object than "pet" in this example.

Questions. To identify targets, one can ask a search question in the form "Given a query vertex q in tree $\mathcal{H}$, can vertex q reach one of targets in $\mathcal{T}$?". Formally,

Definition 1 (Questions). *Given a query vertex* q *and targets* $\mathcal{T}$ *in tree* $\mathcal{H}$*, the search question is defined as* $\mathsf{reach}(q)$. *The boolean answer of* $\mathsf{reach}(q)$ *is either* Yes *or* No.
If $\mathsf{reach}(q) =$ Yes, *then* $\exists t \in \mathcal{T}$ *such that* $q \longrightarrow t$;
Otherwise, $\mathsf{reach}(q) =$ No, *i.e.,* $\forall t \in \mathcal{T}$, $q \not\rightarrow t$.

For example, in Figure 1(a), the question "Can the vertex labeled 'bear' reach one of targets in $\mathcal{T}$?" will get the No answer. None of "bear", "black bear", and "brown bear" will be the correct label. On the contrary, the question "Can the vertex labeled 'pet' reach one of targets in $\mathcal{T}$?" will get the Yes answer. One-shot question asking is limited to figure out where the targets are in a large tree $\mathcal{H}$. One can interactively ask more questions to identify the targets accurately. However, in our kBM-IGS setting, we are given a budget b for the number of questions that can be asked. This is because asking questions is usually costly in real applications, e.g., on Mechanical Turk [1]. It is also not practical to ask users numerous questions as they may not be willing to answer too many questions. After b rounds of question-asking, one finally makes a decision to choose the answers to represent the targets.

Yes-candidates. We say that a vertex v is a Yes-candidate for targets $\mathcal{T}$ if and only if $\exists t \in \mathcal{T}$ such that $v \rightarrow t$. Obviously, the root r is always a Yes-candidate for any targets $\mathcal{T}$. We define the Yes-candidates as the set of Yes-candidate for targets $\mathcal{T}$ as follows.

Definition 2 (Yes-candidates). *Given the targets $\mathcal{T}$ in tree $\mathcal{H}$, and several rounds of asking questions $Q = \{q_0, q_1, ..., q_l\}$ where l is a positive integer and $q_0 = r$, the* Yes-candidates *are defined as*

$$\mathcal{Y} = \bigcup_{q_i \in Q, \mathsf{reach}(q_i) = \mathsf{Yes}} \mathsf{anc}(q_i).$$

Example 2. *Assume that the targets $\mathcal{T} = \{v_2, v_8\}$ in Figure 3 and the questions $Q = \{v_0, v_2, v_3, v_5\}$. The answers are* $\mathsf{reach}(v_2) =$ Yes, $\mathsf{reach}(v_3) =$ Yes, *and* $\mathsf{reach}(v_5) =$ No, *thus* $\mathcal{Y} = \{v_0, v_1, v_2, v_3\}$.

Selections. The selections, denoted as $\mathcal{S}$, are a subset of Yes candidates $\mathcal{Y}$, which are selected by the algorithms to match the targets $\mathcal{T}$ as closely as possible. For example, in Figure 1, assume that we have questioned "pet" and get the Yes answer. We can select "animal", "domestication", or "pet" because they must be the correct label.

Overall, the goal of kBM-IGS interactive scheme is to use a few questions to determine the selections $\mathcal{S} \subseteq \mathcal{Y}$ to approach the targets $\mathcal{T}$ as closely as possible.

3.3 Penalty between Selections and Targets

Given a budget of questions that can be asked and an unknown number of targets, it is challenging to determine the locations of targets in a large tree $\mathcal{H}$. Instead of giving a simple boolean result, we develop a metric to quantify the goodness of our selections. In the following, we introduce another important feature of *penalty* in kBM-IGS. The penalty is an evaluation metric defined on the basis of distance, which measures the closeness between $\mathcal{S}$ and $\mathcal{T}$.

Pair-wise Penalty. Assume that we use a vertex $v \in V$ to cover a given target $t \in \mathcal{T}$. If $v = t$, the choice v exactly identifies the target t. If $v \neq t$, it needs to give a penalty score for using v to cover the target t. Hence, we give a definition of pair-wise penalty score

$$\mathsf{f}\langle v, t\rangle = \begin{cases} \mathsf{dist}\langle v, t\rangle, & \text{if } v \in \mathsf{anc}(t) \\ \mathsf{dist}\langle r, t\rangle, & \text{if } v \notin \mathsf{anc}(t) \end{cases} \quad (1)$$

By the above definitions, we consider two cases: 1) $v \in \mathsf{anc}(t)$ and 2) $v \notin \mathsf{anc}(t)$. First, for $v \in \mathsf{anc}(t)$, indicating $v \rightarrow t$, the best selection v should have $\mathsf{dist}\langle v, t\rangle = 0$. The further the distance $\mathsf{dist}\langle v, t\rangle$, the larger the penalty. Second, for $v \notin \mathsf{anc}(t)$, indicating $v \nrightarrow t$, we give a full penalty of the largest distance between r and t, for using v to cover t, i.e., $\mathsf{f}\langle v, t\rangle = \mathsf{dist}\langle r, t\rangle$. The deeper the location of target t, the larger the penalty. As a result, if a target can be reached by our selections, we use the shortest distance to indicate its closeness. Otherwise, if a target is not reachable from our selections, we give a distance-based penalty.

Set-wise Penalty. Based on pair-wise penalty, we give the definitions of set-wise penalty distance below.

Definition 3 (Penalty). *Given a set of targets $\mathcal{T}$ and a set of selections $\mathcal{S}$, the penalty of $\mathcal{S}$ covering a target $t \in \mathcal{T}$ is defined as the minimum penalty of using a vertex $v \in \mathcal{S}$ to cover t, denoted as*

$$\mathsf{f}(\mathcal{S}, t) = \min_{v \in \mathcal{S}} \mathsf{f}\langle v, t\rangle = \min_{v \in \mathcal{S} \cup \{r\}} \mathsf{dist}\langle v, t\rangle. \quad (2)$$

Moreover, the penalty of $\mathcal{S}$ covering targets $\mathcal{T}$ is defined as the total penalty sum of $\mathcal{S}$ covering all targets $t \in \mathcal{T}$, denoted by

$$\mathsf{f}(\mathcal{S}, \mathcal{T}) = \sum_{t \in \mathcal{T}} \mathsf{f}(\mathcal{S}, t) = \sum_{t \in \mathcal{T}} \min_{v \in \mathcal{S} \cup \{r\}} \mathsf{dist}\langle v, t\rangle. \quad (3)$$

Obviously, if $\mathcal{S} = \mathcal{T}$, the penalty is $\mathsf{f}(\mathcal{S}, \mathcal{T}) = 0$. The smaller the penalty, the better the selections $\mathcal{S}$. In Figure 3, assume that $\mathcal{S} = \{v_2, v_3\}$ and $\mathcal{T} = \{v_2, v_5, v_8\}$, thus $\mathsf{f}\langle v_2, v_8\rangle = 3$ and $\mathsf{f}\langle v_3, v_8\rangle = 1$. The set-wise penalty $\mathsf{f}(\mathcal{S}, v_8) = 1$, $\mathsf{f}(\mathcal{S}, v_5) = 2$ and $\mathsf{f}(\mathcal{S}, \mathcal{T}) = 3$.

3.4 Problem Formulation

We formulate the problem of budget constrained interactive graph search for multiple targets (kBM-IGS) as follows.

Problem 1 (kBM-IGS problem). *Given a hierarchical directed tree $\mathcal{H} = (V, E)$ rooted at r, a target set $\mathcal{T} \subseteq V$, a budget of $b \geq 1$ questions that can be asked, and a positive integer k, the problem is asking b questions $Q = \{q_0, q_1, ..., q_b\}$* one by one *to determine a non-empty set of selections $\mathcal{S}^* \subseteq \mathcal{Y}$ such that $|\mathcal{S}^*| \leq k$ and the penalty $\mathsf{f}(\mathcal{S}^*, \mathcal{T})$ is the smallest. Equivalently,*

$$\mathcal{S}^* = \arg \min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{T})$$

$$s.t., \mathcal{Y} = \bigcup_{q_i \in Q, \mathsf{reach}(q_i) = \mathsf{Yes}} \mathsf{anc}(q_i).$$

Note that the maximum number of selections k where $k \geq |\mathcal{S}|$, could be either larger or smaller than $|\mathcal{T}|$ as we do not know the number of targets $\mathcal{T}$ in real applications. For the example in Figure 1 with $k = 2$ and $b = 5$, assume that we get Yes answers for questions "pet" and "fish" and No answers for questions "wild", "shell", and "whale". The best selections are $\mathcal{S}^*$ ={"pet", "fish"}.

3.5 Applications

We motivate the kBM-IGS problem with two useful applications.

Image categorization. New images (e.g., biomedical images, surveillance photos, and user-uploaded images in online social networks) are continuously being generated and need to be classified by humans to identify objects and labels [23, 29]. Our kBM-IGS scheme can leverage the crowd-aided intelligence to identify multiple objects in an image using a budget constrained interactive graph search. First, an image may have multiple labels, e.g., the image shown in Figure 1(b) has two labels "cat" and "fish". Second, answering a question involves certain communication, latency, and monetary costs. Given a limited budget for rewards, it is necessary to constrain the total number of questions to be asked and select the most suitable labels to categorize the image.

Cold-start recommendation. Due to the lack of users' preferences in cold-start recommendations, online platforms (e.g., Twitter, TikTok, and YouTube) can ask a few questions to identify users' interests and then offer personalized recommendations in a more effective way. Users' preferences may be diverse, which are usually not limited to a single interest, e.g., one may like "traveling", "financial news", "movies", and so on. To avoid users becoming bored due to being asked too many questions, our kBM-IGS scheme can ask only a small number of questions using an interest hierarchy and adjust asking strategy dynamically based on the previous answers.

Remark. In practical crowdsourcing applications, while human mistakes are inevitable, they can be minimized or eliminated by adopting effective quality control measures such as expert review, majority voting, group consensus, and so on [9]. As validated in [29], the influence of such mistakes on the outcome of the graph search algorithms is negligible. Thus, as with the previous works [17, 23, 29], we assume in our algorithm design that the workers always give correct answers. For those cases where human mistakes are not eliminated and the workers give wrong answers, we will assess the quality of our methods in Section 7.

4 THE PROPOSED FRAMEWORK

In this section, we analyze the properties of kBM-IGS problem and briefly introduce our algorithmic framework.

4.1 Theoretical Analysis

We first give new definitions of *potential targets* and then analyze the relationships between *questions* and *potential targets*.

Potential targets. We first define the potential targets, denoted by $\mathcal{P}$, as a candidate set of vertices that could be exact targets of $\mathcal{T}$ where $\mathcal{T} \subseteq \mathcal{P}$. Obviously, if no question has been asked, every vertex $v \in V$ could be a potential target due to the limited prior information, i.e., $\mathcal{P} = V$. However, as more questions are asked, the potential targets could decrease as some vertices may be pruned from $\mathcal{P}$ for violating the target constraints, no matter whether the question answer is Yes or No. We have the following lemmas.

Lemma 1. *Given a vertex $q \in V$, if the question* $\mathsf{reach}(q) = \mathsf{No}$, *all vertices $u \in \mathsf{des}(q)$ are not targets, which should be pruned from potential targets, i.e.,* $\mathsf{des}(q) \cap \mathcal{P} = \emptyset$.

Proof. First, for $\mathsf{reach}(q) = \mathsf{No}$, q cannot reach any target $t \in \mathcal{T}$. For each vertex $u \in \mathsf{des}(q)$, u also cannot reach any target $t \in \mathcal{T}$, $u \notin \mathcal{T}$. Thus, $\mathsf{des}(q) \cap \mathcal{T} = \emptyset$, and all vertices $\mathsf{des}(q)$ can be pruned from potential targets, denoted as $\mathsf{des}(q) \cap \mathcal{P} = \emptyset$. □

Lemma 2. *Given a vertex $q \in V$, if the question* $\mathsf{reach}(q) = \mathsf{Yes}$, *all vertices $u \in \mathsf{anc}(q) \setminus \{q\}$ are not targets, which should be pruned from potential targets, i.e.,* $\mathsf{anc}(q) \cap \mathcal{P} = \{q\}$.

Proof. For $\mathsf{reach}(q) = \mathsf{Yes}$, $\exists t \in \mathcal{T}$ satisfies $q \rightarrow t$. By the independence property of targets, for other target $t' \in \mathcal{T}, t' \neq t$, we can get $t' \nrightarrow t$. Thus, for the vertex $u \in \mathsf{anc}(q) \setminus \{q\}$, $u \longrightarrow t$, so $u \notin \mathcal{T}$ and all vertices u can be pruned from potential targets. □

For example, if we question the vertex v_3 in Figure 3 and get the No answer, the vertices v_3, v_6, v_7, v_8 will be pruned from $\mathcal{P}$. Similarly, if we get the Yes answer, the vertices v_0, v_1 will be pruned.

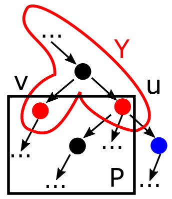


Figure 4: An example of Yes-candidates **and potential targets. The red vertices get** Yes **answer, the blue vertices get** No **answers and the black vertices are not questioned. The red area is the** Yes-candidates **and the black area is the potential targets.**

Properties of Yes-candidates and potential targets. Next, we analyze the properties of the Yes-candidates and potential targets.

Lemma 3. *$\mathcal{Y} \cap \mathcal{P} \neq \emptyset$ always holds.*

Proof. A complete proof is reported in the article [35]. □

Lemma 4. *For a vertex $t \in V$ is a target, $t \in \mathcal{T}$, if and only if* $\mathsf{reach}(t) = \mathsf{Yes}$ *and* $\mathsf{reach}(u) = \mathsf{No}$ *for all $u \in \mathsf{child}(t)$.*

Proof. A complete proof is available in [35]. □

Theorem 1. *If $\mathcal{P} \subseteq \mathcal{Y}$, the targets are exactly as $\mathcal{T} = \mathcal{P}$.*

Proof. For $\forall t \in \mathcal{P}$, $\mathsf{reach}(t) = \mathsf{Yes}$. By Lemma 2, $\mathsf{reach}(v) = \mathsf{No}$ holds for $v \in \mathsf{child}(t)$. Moreover, by Lemma 4, $\mathcal{P} \subseteq \mathcal{T}$. As the definition of potential targets $\mathcal{T} \subseteq \mathcal{P}$, thus $\mathcal{P} = \mathcal{T}$. □

Figure 4 shows an example of Yes-candidates and potential targets. $\mathcal{P} \cap \mathcal{Y} \neq \emptyset$. If the children of u and v are all questioned and get the No answer, $\mathcal{P} = \{u, v\} \subseteq \mathcal{Y}$ and the targets will be $\mathcal{T} = \{u, v\}$.

4.2 kBM-IGS Framework

In this section, we introduce a novel kBM-IGS framework for identifying multiple targets via a series of b interactive questions. The key idea is asking *good questions* to reduce potential targets $\mathcal{P}$ and refine $\mathcal{Y}$ to be specified by Theorem 1.

Motivations. We use a toy example $\mathcal{H}$ in Figure 3 to show the general ideas of our framework. Assume that $\mathcal{P} = V$ and $\mathcal{Y} = \{r\}$. First, we consider a vertex v_1 and ask the question $\mathsf{reach}(v_1)$. If $\mathsf{reach}(v_1) = \mathsf{No}$, all the descendants of v_1 can be pruned from $\mathcal{P}$, which achieves a considerable gain by reducing $|\mathcal{P}|$ from 10 to 2. But, if $\mathsf{reach}(v_1) = \mathsf{Yes}$, $\mathcal{P}$ will only reduce v_0, which achieves a limited gain. Unfortunately, assuming that the targets are randomly distributed in V, v_1 has a low probability of getting $\mathsf{reach}(v_1) = \mathsf{No}$. This is because there exist 8 descendants of v_1, and if any vertex $u \in \mathsf{des}(v_1)$ is the target, $\mathsf{reach}(v_1) = \mathsf{Yes}$ holds. Thus, v_1 may not be a good choice for questioning. Second, we consider a leaf vertex v_9. If $\mathsf{reach}(v_9) = \mathsf{Yes}$, we surely know that v_9 is one desired target, i.e., $v_9 \in \mathcal{T}$, which achieves lots of gains. But, if $\mathsf{reach}(v_9) = \mathsf{No}$, $\mathcal{P}$ will only reduce v_9, which achieves a limited gain. However, it has a high probability of getting $\mathsf{reach}(v_9) = \mathsf{No}$ and a low probability to $\mathsf{reach}(v_9) = \mathsf{Yes}$. We need to select good vertices by making a balanced trade-off between the probability and gains. To do so, the kBM-IGS framework develops a ranking evaluation function for vertices, which is based on *target probability* and *gain score*.

Algorithm 1 kBM-IGS Framework

```
Input: A hierarchy tree $\mathcal{H} = (V, E)$, a budget $b$, a number $k$.
Output: Selections $\mathcal{S}$ with $|\mathcal{S}| \leq k$.
1: Let $\mathcal{Y} \leftarrow \{r\}$, $\mathcal{P} \leftarrow V$;
2: Initialize the probability $\mathsf{pr}(v)$ for every vertex $v \in V$;
3: for $i \leftarrow 1$ to $b$ do
4:     for $v \in \mathcal{P} \setminus \mathcal{Y}$ do
5:         Calculate $\mathsf{p}_{\mathsf{Yes}}(v), \mathsf{p}_{\mathsf{No}}(v), \mathsf{g}_{\mathsf{Yes}}(v), \mathsf{g}_{\mathsf{No}}(v)$;
6:         $\mathsf{Gain}(v) \leftarrow \mathsf{g}_{\mathsf{Yes}}(v) \cdot \mathsf{p}_{\mathsf{Yes}}(v) + \mathsf{g}_{\mathsf{No}}(v) \cdot \mathsf{p}_{\mathsf{No}}(v)$ by Def. 6;
7:     $q_i \leftarrow \arg\max_{v \in \mathcal{P} \setminus \mathcal{Y}} \mathsf{Gain}(v)$;
8:     Ask the question $\mathsf{reach}(q_i)$;
9:     if $\mathsf{reach}(q_i) = \mathsf{Yes}$ then
10:        $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathsf{anc}(q_i)$ by Def. 1;
11:    Update the potential candidates $\mathcal{P}$ and vertex probabilities $\mathsf{pr}$ accordingly if needed;
12:    if $\mathcal{P} \subseteq \mathcal{Y}$ then break by Theorem. 1;
13: $\mathcal{S}^* \leftarrow \arg\min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$;
14: return $\mathcal{S}^*$;
```

Target probability. For a vertex v, we denote the target probabilities of $\mathsf{reach}(v) = \mathsf{Yes}$ and $\mathsf{reach}(v) = \mathsf{No}$ respectively as $\mathsf{p}_{\mathsf{Yes}}(v)$ and $\mathsf{p}_{\mathsf{No}}(v)$, which satisfy $\mathsf{p}_{\mathsf{Yes}}(v) + \mathsf{p}_{\mathsf{No}}(v) = 1$. The specific calculations of $\mathsf{p}_{\mathsf{Yes}}(v)$ and $\mathsf{p}_{\mathsf{No}}(v)$ are based on the descendants $\mathsf{des}(v)$, which will be introduced in Sections 5 and 6.

Gain score. We first define the potential penalty. Instead of using the targets $\mathcal{T}$ as in Problem 1, we define the potential penalty to measure the minimum distance between feasible selections $\mathcal{S}$ and potential targets $\mathcal{P}$ as we do not know the exact $\mathcal{T}$, as follows.

Definition 4 (Potential Penalty). *The potential penalty is denoted as* $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k) = \min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$.

The potential penalty $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ is to select the best k vertices from the Yes-candidates $\mathcal{Y}$ in order to identify the potential targets in $\mathcal{P}$. The less $|\mathcal{P}|$ and the closer $\mathcal{S}$ to $\mathcal{P}$ is, the lower score $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ is, which is better. Next, we present the definition of gain score. For a given vertex $v \in V$ with existing $\mathcal{P}$ and $\mathcal{Y}$, we ask a new question $\mathsf{reach}(v)$, and present two gain scores for the different answers $\mathsf{reach}(v) = \mathsf{Yes}$ and $\mathsf{reach}(v) = \mathsf{No}$ respectively, as follows.

Definition 5 (Yes & No Gains). *The gain of* $\mathsf{reach}(v) = \mathsf{Yes}$ *is denoted as* $\mathsf{g}_{\mathsf{Yes}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{g}(\hat{\mathcal{Y}}_v, \hat{\mathcal{P}}_v, k)$ *where* $\hat{\mathcal{Y}}_v$, $\hat{\mathcal{P}}_v$ *are the updated potential targets and* Yes-candidates *after asking the question* $\mathsf{reach}(v) = \mathsf{Yes}$*; Similarly, the gain of* $\mathsf{reach}(v) = \mathsf{No}$ *is denoted as* $\mathsf{g}_{\mathsf{No}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{g}(\bar{\mathcal{Y}}_v, \bar{\mathcal{P}}_v, k)$ *where* $\bar{\mathcal{Y}}_v$, $\bar{\mathcal{P}}_v$ *are the updated potential targets and* Yes-candidates *after asking the question* $\mathsf{reach}(v) = \mathsf{No}$.

Based on the target probabilities and gain scores, we define an integrated function of expected gain as follows.

Definition 6 (Expected Gain). *Given a vertex* v *in* $\mathcal{H}$*, the expected gain of asking the question* $\mathsf{reach}(v)$ *is denoted as*

$$\mathsf{Gain}(v) = \mathsf{g}_{\mathsf{Yes}}(v) \cdot \mathsf{p}_{\mathsf{Yes}}(v) + \mathsf{g}_{\mathsf{No}}(v) \cdot \mathsf{p}_{\mathsf{No}}(v).$$

The larger the expected gain, the better the choice for questioning.

Algorithm. The algorithm of kBM-IGS framework is outlined in Algorithm 1. The general idea is to use a greedy strategy to select the vertex with the largest expected gain at each round of question-asking. The framework has an input of a hierarchy tree $\mathcal{H}$, a budget of b questions that can be asked, and a number k. First, it initializes the Yes-candidates $\mathcal{Y}$ as $\{r\}$ and the potential targets $\mathcal{P}$ as the whole vertex set V (line 1). Note that if the vertices have no probabilities, we can set all vertices to have the same probability as $\frac{k}{n}$ (line 2). The algorithm then iteratively selects one best vertex $q_i \in \mathcal{P} \setminus \mathcal{Y}$ and asks the question $\mathsf{reach}(q_i)$ until the quota of b questions is used up (lines 3-12). For each round, it calculates the target probabilities of $\mathsf{p}_{\mathsf{Yes}}(v)$, $\mathsf{p}_{\mathsf{No}}(v)$ and the Yes &No gains of $\mathsf{g}_{\mathsf{Yes}}(v)$, $\mathsf{g}_{\mathsf{No}}(v)$ for each vertex $v \in \mathcal{P} \setminus \mathcal{Y}$. Then, the expected gains of all vertices are computed (lines 4-6). The algorithm next finds the vertex q_i with the largest expected gain and asks the question (lines 7-8). According to the answer, the Yes-candidates, potential targets, and vertex probabilities are updated in accordance with the answer (lines 9-11). Finally, after b questions or the identification of exact targets, the algorithm selects the best selection $\mathcal{S}^* = \arg\min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$ and return $\mathcal{S}^*$ as the final selections (lines 13-14).

5 SINGLE TARGET SEARCH

In this section, we investigate one special case of kBM-IGS problem, i.e., the SingleTarget problem [17, 29], where $|\mathcal{T}| = 1$. On the basis of the kBM-IGS framework, we develop a STBIS method to identify one vertex as $\mathcal{S}$ by asking b questions.

5.1 Single Target Problem Analysis

As an instance problem, the SingleTarget problem inherits all properties of kBM-IGS described in Section 4 and enjoys its own properties. Assume that the initial $\mathcal{P} = V$ and $\mathcal{Y} = \{r\}$, and the target $\mathcal{T} = \{t\}$. We can ask a question and interactively update $\mathcal{P}$ as $\mathcal{P}_{new}$ and $\mathcal{Y}$ as $\mathcal{Y}_{new}$ by obeying the two following rules.

First, as $|\mathcal{T}| = 1$, for each question, the best strategy is to ask a vertex v that is a potential target $v \in \mathcal{P} \setminus \mathcal{Y}$. Otherwise, we consider two cases. First, if we ask a vertex $v \in \mathcal{Y}$, the answer of $\mathsf{reach}(v)$ is always Yes; Second, if we ask a vertex $v \in V \setminus (\mathcal{P} \cup \mathcal{Y})$, the answer of $\mathsf{reach}(v)$ is always No. Thus, it achieves no benefit gain but wastes one question from the budget. Moreover, in contrast to Lemma 2, in SingleTarget problem, more vertices can be pruned after Yes answer as follows.

Lemma 5. *For a vertex* $q \in \mathcal{P} \setminus \mathcal{Y}$*, if* $\mathsf{reach}(q) = \mathsf{Yes}$*, none of* $u \in \mathcal{P} \setminus \mathsf{des}(q)$ *are potential targets and we update* $\mathcal{P}_{new} = \mathsf{des}(q) \cap \mathcal{P}$.

Proof. A complete proof is available in [35]. □

Second, the Yes-candidates can be updated only when a question $\mathsf{reach}(v) = \mathsf{Yes}$ by Def. 1, where the updated Yes-candidates $\mathcal{Y}_{new} = \mathcal{Y} \cup \mathsf{anc}(v)$. However, $\mathcal{Y} \subseteq \mathcal{Y}_{new} \subseteq \mathsf{anc}(t)$ always holds, i.e., all Yes-candidates lie along the path from root r to target t. The penalty function $\mathsf{f}(\mathcal{S}, \mathcal{T})$ in Def. 3 tells us that keeping one vertex $s \in \mathcal{Y}_{new}$ closest to t is enough. In other words, it achieves the minimum penalty $\mathsf{f}(\mathcal{S}, \mathcal{T}) = \mathsf{f}(\{s\}, \{t\})$. Thus, $\mathcal{Y}_{new}$ can be updated as $\mathcal{Y}_{new} = \{s\}$, where s has the largest depth $\mathsf{dist}\langle r, s\rangle$ in $\mathcal{H}$ and the question $\mathsf{reach}(s) = \mathsf{Yes}$.

Lemma 6. *For $\mathcal{Y} = \{s\}$, both $\mathcal{Y} \cap \mathcal{P} = \{s\}$ and the penalty $\mathrm{g}(\mathcal{Y}, \mathcal{P}, 1) = \mathrm{f}(\{s\}, \mathcal{P})$ hold.*

Proof. First, we prove $s \in \mathcal{P}$. Since $s \in \mathcal{Y}$, s will not be pruned from No questions according to Lemma 1. Furthermore, as s is the deepest vertex in $\mathcal{Y}$, s will not be pruned from Yes questions according to Lemma 5. So, $s \in \mathcal{P}$ and $\mathcal{Y} \cap \mathcal{P} = \{s\}$. Moreover, since $|\mathcal{Y}| = |\{s\}| = 1$, $\mathrm{g}(\mathcal{Y}, \mathcal{P}, 1) = \mathrm{f}(\{s\}, \mathcal{P})$. □

Based on the above properties, we have two useful updating rules.

Rule 1. *For a vertex $v \in \mathcal{P}$ with* $\mathrm{reach}(v) = \mathrm{Yes}$*, we update the potential targets $\mathcal{P}_{new} = \mathrm{des}(v) \cap \mathcal{P}$ and the* Yes*-candidates $\mathcal{Y}_{new} = \{v\}$.*

Rule 2. *For a vertex $v \in \mathcal{P}$ with* $\mathrm{reach}(v) = \mathrm{No}$*, we update the potential targets $\mathcal{P}_{new} = \mathcal{P} \setminus \mathrm{des}(v)$ and keep the* Yes*-candidates unchanged $\mathcal{Y}_{new} = \mathcal{Y} = \{s\}$.*

5.2 The STBIS Algorithm

Probability calculation. Before asking any questions, each vertex has an equal probability of being the target. Thus, we let each vertex u have a probability of $\mathrm{pr}(u) = \frac{1}{n}$ where $n = |V|$. Our proposed algorithm can be easily extended to other vertex probability distribution based on historical query logs as [17]. Therefore, for a vertex v, the target probability for each vertex v follows $\mathrm{p}_{\mathrm{Yes}}(v) = \sum_{u \in \mathrm{des}(v)} \mathrm{pr}(u)$ and the no probability follows $\mathrm{p}_{\mathrm{No}}(v) = 1 - \mathrm{p}_{\mathrm{Yes}}(v)$. As more question answers are discovered, the vertex probabilities need to be updated accordingly. The updated probability after each question is calculated as:

$$\mathrm{pr}(u) = \begin{cases} \mathrm{pr}(u) \cdot \dfrac{|\mathcal{P}|}{|\mathcal{P}_{new}|}, & u \in \mathcal{P}_{new} \\ 0, & u \notin \mathcal{P}_{new} \end{cases} \tag{4}$$

where $\mathcal{P}_{new}$ is the potential targets after asking a question $\mathrm{reach}(q_i)$ where $1 \le i \le b$. The general idea is to assign impossible targets with a probability value of zero and keep the sum probability as 1.

STBIS algorithm. The detailed procedure of STBIS is outlined in Algorithm 2, which finds the vertices with the largest gain to ask interactive questions and finally identifies a selection to represent the target within b questions. The algorithm first initializes the Yes-candidates $\mathcal{Y}$ as a root r and potential targets $\mathcal{P} = V$ (line 1), and uniformly assigns the vertex probability (line 2). Then, it calculates the Yes&No probability (line 5), the Yes&No gain scores (lines 6-9), and the expected gain $\mathrm{Gain}(v)$ (line 10) for all potential targets $v \in \mathcal{P}$. Next, the algorithm chooses a vertex $q_i \in \mathcal{P}$ with the largest expected gain and ask question $\mathrm{reach}(q_i)$ (lines 11-12). If $\mathrm{reach}(q_i) = \mathrm{Yes}$, it updates $\mathcal{P}_{new} = \mathrm{des}(q_i)$ and $\mathcal{Y} = \{q_i\}$ by Rule 1 (lines 13-14); Otherwise, if $\mathrm{reach}(q_i) = \mathrm{No}$, it updates $\mathcal{P}_{new} = \mathcal{P} \setminus \mathrm{des}(q_i)$ by Rule 2 (lines 15-16). It updates the probability using Eq. 4 (lines 17-18), and assign $\mathcal{P} = \mathcal{P}_{new}$. Finally, the algorithm returns the vertex $s \in \mathcal{Y}$ as the selection (line 21) and terminates early if $\mathcal{P} = \mathcal{Y}$ by Theorem 1 (line 20).

Algorithm 2 STBIS

```
Input: A hierarchy tree $\mathcal{H} = (V, E)$, root $r$, budget $b$, and $k = 1$.
Output: One selection $s$.
 1: Let $\mathcal{Y} \leftarrow \{r\}$, $\mathcal{P} \leftarrow V$;
 2: Assign the probability $\mathrm{pr}(v) = 1/n$ for $v \in V$;
 3: for $i \leftarrow 1$ to $b$ do
 4:     for $v \in \mathcal{P} \setminus \mathcal{Y}$ do
 5:         $\mathrm{p}_{\mathrm{Yes}}(v) \leftarrow \sum_{u \in \mathrm{des}(v)} \mathrm{pr}(u)$, $\mathrm{p}_{\mathrm{No}}(v) \leftarrow 1 - \mathrm{p}_{\mathrm{Yes}}(v)$;
 6:         Calculate $\hat{\mathcal{P}}_v$ as $\mathcal{P}_{new} \leftarrow \mathcal{P} \cap \mathrm{des}(v)$ by Rule 1;
 7:         Update $\mathrm{g}_{\mathrm{Yes}}(v) \leftarrow \mathrm{f}(\mathcal{Y}, \mathcal{P}) - \mathrm{f}(\{v\}, \mathcal{P}_{new})$;
 8:         Calculate $\bar{\mathcal{P}}_v$ as $\mathcal{P}_{new} \leftarrow \mathcal{P} \setminus \mathrm{des}(v)$ by Rule 2;
 9:         Update $\mathrm{g}_{\mathrm{No}}(v) \leftarrow \mathrm{f}(\mathcal{Y}, \mathcal{P}) - \mathrm{f}(\mathcal{Y}, \mathcal{P}_{new})$;
10:         $\mathrm{Gain}(v) \leftarrow \mathrm{g}_{\mathrm{Yes}}(v) \cdot \mathrm{p}_{\mathrm{Yes}}(v) + \mathrm{g}_{\mathrm{No}}(v) \cdot \mathrm{p}_{\mathrm{No}}(v)$;
11:     $q_i \leftarrow \arg\max_{v \in \mathcal{P} \setminus \mathcal{Y}} \mathrm{Gain}(v)$;
12:     Ask the question $\mathrm{reach}(q_i)$;
13:     if $\mathrm{reach}(q_i) = \mathrm{Yes}$ then
14:         $\mathcal{P}_{new} \leftarrow \mathcal{P} \cap \mathrm{des}(q_i)$; $\mathcal{Y} \leftarrow \{q_i\}$;
15:     else
16:         $\mathcal{P}_{new} \leftarrow \mathcal{P} \setminus \mathrm{des}(q_i)$; $\mathcal{Y}$ keeps unchanged;
17:     Update $\mathrm{pr}(u) = 0$ for $u \in \mathcal{P} \setminus \mathcal{P}_{new}$;
18:     Update $\mathrm{pr}(u) = \mathrm{pr}(u) \cdot \frac{|\mathcal{P}|}{|\mathcal{P}_{new}|}$ for $u \in \mathcal{P}_{new}$;
19:     $\mathcal{P} \leftarrow \mathcal{P}_{new}$;
20:     if $\mathcal{P} = \mathcal{Y}$ then return $s \in \mathcal{Y}$;
21: return $s \in \mathcal{Y}$;
```

Table 2: The values of $\mathrm{g}_{\mathrm{Yes}}$, g_{No}, $\mathrm{p}_{\mathrm{Yes}}$, and p_{No} of first question and the Gain in two questions. Here, $b = 2$ and $\mathcal{T} = \{v_5\}$.

Node	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
$\mathrm{g}_{\mathrm{Yes}}$	9	20	17	20	19	20	20	20	20
g_{No}	19	1	11	2	5	3	3	3	3
$\mathrm{p}_{\mathrm{Yes}}$	0.8	0.1	0.4	0.1	0.2	0.1	0.1	0.1	0.1
p_{No}	0.2	0.9	0.6	0.9	0.8	0.9	0.9	0.9	0.9
Gain^1	11	2.9	**13.4**	3.8	7.8	4.7	4.7	4.7	4.7
Gain^2	**6**	2.33	/	3.17	**6**	/	/	/	4

Example 3. *Assume that $\mathcal{T} = \{v_5\}$ and $b = 2$ in Figure 3. Table 2 shows the expected gains of all vertices. In the first round, Algorithm 2 questions v_3 and gets the* No *answer. Then, v_3, v_6, v_7, v_8 are pruned. The vertices v_1 and v_5 get the maximum gain in the second round. If v_5 is selected, the penalty is 0. If v_1 is selected, the penalty is 1.*

Complexity analysis. STBIS in Algorithm 2 takes $O(n)$ time to generate a question, which uses a DFS procedure to calculate expected gain $\mathrm{Gain}(v)$ for all vertices v [35]. The overall time complexity of STBIS takes $O(bn)$ time in $O(n)$ space for generating b questions.

6 MULTIPLE TARGETS SEARCH

In this section, we propose efficient algorithms for identifying k selections for multiple targets. We first introduce the probability setting and updating rules for the identification of multiple targets. Then, we propose a kBM-DP method using dynamic programming techniques and improve its efficiency by leveraging the techniques of non-diverse selections and bounded pruning.

6.1 Multiple Targets Scheme

We start by presenting *probability setting* and *updating rules.*

Target probability. As there exist multiple targets with $|\mathcal{T}| \geq 1$, we assume that each vertex has an independent probability of being

Algorithm 3 kBM-DP

```
Input: A hierarchy tree $\mathcal{H} = (V, E)$, a budget $b$, a number $k$.
Output: Selections $\mathcal{S}$ with $|\mathcal{S}| \leq k$.
 1: Let $\mathcal{Y} \leftarrow \{r\}, \mathcal{P} \leftarrow V$;
 2: Assign the probability $\mathsf{pr}(v) = k/n$ for $v \in V$;
 3: for $i \leftarrow 1$ to $b$ do
 4:     for $v \in \mathcal{P} \setminus \mathcal{Y}$ do
 5:         $\mathsf{p_{No}}(v) \leftarrow \prod_{u \in \mathsf{des}(v) \cap \mathcal{P}} (1 - \mathsf{pr}(u))$;
 6:         $\mathsf{p_{Yes}}(v) \leftarrow 1 - \mathsf{p_{No}}(v)$;
 7:     Calculate $\mathsf{Gain}(v)$ for all $v \in \mathcal{P} \setminus \mathcal{Y}$ using Algorithm 4;
 8:     $q_i \leftarrow \arg\max_{v \in \mathcal{P} \setminus \mathcal{Y}} \mathsf{Gain}(v)$;
 9:     Ask the question $\mathsf{reach}(q_i)$;
10:     if $\mathsf{reach}(q_i) = \mathsf{Yes}$ then
11:         $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathsf{anc}(q_i)$ by Rule 3;
12:         $\mathcal{P}_{new} \leftarrow \mathcal{P} \setminus (\mathsf{anc}(q_i) \setminus \{q_i\})$ by Rule 3;
13:     else
14:         $\mathcal{P}_{new} \leftarrow \mathcal{P} \setminus \mathsf{des}(q_i)$ by Rule 4;
15:     Update $\mathsf{pr}(u) = 0$ for $u \in \mathcal{P} \setminus \mathcal{P}_{new}$;
16:     Update $\mathsf{pr}(u) = \mathsf{pr}(u) \cdot \frac{|\mathcal{P}|}{|\mathcal{P}_{new}|}$ for $u \in \mathcal{P}_{new}$;
17:     $\mathcal{P} \leftarrow \mathcal{P}_{new}$;
18:     if $\mathcal{P} \subseteq \mathcal{Y}$ then break;
19: $\mathcal{S}^* \leftarrow \arg\min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$;
20: return $\mathcal{S}^*$;
```

a target. As our problem aims at finding k selections and $|\mathcal{T}|$ is unknown, let $\mathsf{pr}(v)$ be the vertex probability of v and $\mathsf{pr}(v) = \frac{k}{n}$. Thus, the target probability of a vertex v is computed as follows. The probability of $\mathsf{reach}(v) = \mathsf{No}$ is denoted as the No probability $\mathsf{p_{No}}(v) = \prod_{u \in \mathsf{des}(v) \cap \mathcal{P}} (1 - \mathsf{pr}(u))$, representing that none of vertices $u \in \mathsf{des}(v) \cap \mathcal{P}$ is a target. Moreover, we update the target probabilities with more questions asked.

Rules of updating $\mathcal{P}$ and $\mathcal{Y}$. Following Def. 2 and Lemmas 1 and 2, we have the following rules for updating $\mathcal{P}$ and $\mathcal{Y}$.

RULE 3. *For a vertex $v \in \mathcal{P}$ with* $\mathsf{reach}(v) = \mathsf{Yes}$*, we update the potential targets $\mathcal{P}_{new} = \mathcal{P} \setminus (\mathsf{anc}(v) \setminus \{v\})$ and update the* Yes-candidates $\mathcal{Y}_{new} = \mathcal{Y} \cup \mathsf{anc}(v)$.

RULE 4. *For a vertex $v \in \mathcal{P}$ with* $\mathsf{reach}(v) = \mathsf{No}$*, we update the potential targets $\mathcal{P}_{new} = \mathcal{P} \setminus \mathsf{des}(v)$ and keep the* Yes-candidates *unchanged $\mathcal{Y}_{new} = \mathcal{Y}$.*

According to Rules 3 and 4, we compute both $\mathsf{g_{Yes}}(v)$ and $\mathsf{g_{No}}(v)$, which equals $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$− $\mathsf{g}(\mathcal{Y}_{new}, \mathcal{P}_{new}, k)$ by Def. 5. However, as we know that $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k) = \min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$, it is difficult to efficiently compute the penalty $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ with a straightforward enumeration of $\mathcal{S} \subseteq \mathcal{Y}$ for $k > 1$. In the following sections, we mainly focus on developing efficient approaches to compute $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ and update Yes&No gain scores $\mathsf{g_{Yes}}(v)$ and $\mathsf{g_{No}}(v)$.

6.2 kBM-DP Algorithm

In this section, we propose a kBM-DP algorithm for identifying multiple targets based on the kBM-IGS framework in Algorithm 1.

Algorithm 4 kBM-DP: Calculate Expected Gains

```
Input: $\mathcal{H} = (V, E)$, $\mathsf{p_{Yes}}(.)$, $\mathsf{p_{No}}(.)$, $\mathcal{P}$, $\mathcal{Y}$, and $k$.
Output: $\mathsf{Gain}(v)$ for all vertices $u \in \mathcal{P} \setminus \mathcal{Y}$.
Calculate $\mathsf{DP}(u, w, k)$ for all vertices $u \in \mathcal{P}$ and $w \in \mathsf{anc}(u)$ in Eq. 5.
for $v \in \mathcal{P} \setminus \mathcal{Y}$ do
    $\hat{\mathcal{P}} \leftarrow \mathcal{P} \setminus (\mathsf{anc}(v) \setminus \{v\})$;        // $\mathsf{reach}(v) = \mathsf{Yes}$
    $\mathsf{g_{Yes}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{calg_{Yes}}(v, k)$;
    $\bar{\mathcal{P}} \leftarrow \mathcal{P} \setminus \mathsf{des}(v)$;                    // $\mathsf{reach}(v) = \mathsf{No}$
    $\mathsf{g_{No}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{calg_{No}}(v, k)$;
    $\mathsf{Gain}(v) \leftarrow \mathsf{g_{Yes}}(v) \cdot \mathsf{p_{Yes}}(v) + \mathsf{g_{No}}(v) \cdot \mathsf{p_{No}}(v)$ by Def. 6;
procedure $\mathsf{calg_{Yes}}(u, k)$
    for $v \in \mathsf{anc}(u)$ do
        Recalculate $\mathsf{DP}_Y(v, k)$ by Eq. 6;
        for $w \in \mathsf{anc}(v) \setminus \{v\}$ do
            Recalculate $\mathsf{DP}_N(v, w, k)$ by Eq. 7;
            $\mathsf{DP}(v, w, k) \leftarrow \min\{\mathsf{DP}_N(v, w, k), \mathsf{DP}_Y(v, k)\}$;
    $\mathsf{g}(\hat{\mathcal{Y}}, \hat{\mathcal{P}}, k) \leftarrow \mathsf{DP}(r, r, k)$;
    return $\mathsf{g}(\hat{\mathcal{Y}}, \hat{\mathcal{P}}, k)$;
procedure $\mathsf{calg_{No}}(u, k)$
    for $v \in \mathsf{anc}(u)$ do
        for $w \in \mathsf{anc}(v) \cap \mathcal{Y} \setminus \{v\}$ do
            Recalculate $\mathsf{DP}_N(v, w, k)$ by Eq. 7;
        if $v \in \mathcal{Y}$ then
            Recalculate $\mathsf{DP}_Y(v, k)$ by Eq. 6;
            $\mathsf{DP}(v, w, k) \leftarrow \min\{\mathsf{DP}_N(v, w, k), \mathsf{DP}_Y(v, k)\}$;
        else
            $\mathsf{DP}(v, w, k) \leftarrow \mathsf{DP}_N(v, w, k)$;
    $\mathsf{g}(\bar{\mathcal{Y}}, \bar{\mathcal{P}}, k) \leftarrow \mathsf{DP}(r, r, k)$;
    return $\mathsf{g}(\bar{\mathcal{Y}}, \bar{\mathcal{P}}, k)$;
```

The kBM-DP algorithm. The algorithm of kBM-DP is presented in Algorithm 3. The algorithm first initializes the Yes-candidates $\mathcal{Y}$, the potential targets $\mathcal{P}$, and the independent probability $\mathsf{pr}(v) = \frac{k}{n}$ for each vertex $v \in V$ (lines 1-2). Then, it iteratively selects one best vertex $v \in \mathcal{P} \setminus \mathcal{Y}$ with the largest $\mathsf{Gain}(v)$ and asks question $\mathsf{reach}(v)$ until all b questions have been asked (lines 3-18). At the i-th round of asking question, it updates the target probabilities for all vertices $\mathcal{P} \setminus \mathcal{Y}$ (lines 4-6). The algorithm invokes Algorithm 4 to calculate all expected gains (line 7). Next, the algorithm asks question $\mathsf{reach}(q_i)$, and updates the vertex probabilities, $\mathcal{P}$ and $\mathcal{Y}$ accordingly by Rules 3 and 4 (lines 10-16). Finally, the algorithm returns the selections $\mathcal{S}^* = \arg\min_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \mathsf{f}(\mathcal{S}, \mathcal{P})$ (line 19).

In the following, we introduce a dynamic programming algorithm for calculating the expected gains in Algorithm 4.

Computing $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$. An intuitive approach enumerates all k-sized selections $\mathcal{S} \subseteq \mathcal{Y}$ for finding the best selections S^*, which is inefficient. Thus, we use a dynamic programming technique to calculate the minimum $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ efficiently. The general idea is to divide the global calculation into sub-problems of finding $k' \leq k$ selection vertices optimally in a subtree T_u rooted by a vertex u. The vertex $w \in \mathcal{S} \cap \mathsf{anc}(u)$ is a selection closest to u. Note that if $\mathcal{S} \cap \mathsf{anc}(u) = \emptyset$, we consider $w = r$. Obviously, let $u = r$, $w = r$ and $k' = k$, this subproblem is the same as the best selection of S. Thus, our objective is to calculate it from the sub-problems. We consider two cases of whether we select vertex u or not for each subtree T_u. On one hand, if $u \in \mathcal{Y}$ and we select vertex u into the selections $\mathcal{S}$, for each child node $v_1, v_2..., v_x \in \mathsf{child}(u) \cap \mathcal{P}$, the sub-problem is how to find additional k_x optimal vertices in the subtrees rooted by v_x with the closest selected vertex u and $\sum k_x \leq k' - 1$; On the other hand, if we do not select vertex u into the answer $\mathcal{S}$, for each child node $v_1, v_2..., v_x \in \mathsf{child}(u) \cap \mathcal{P}$, the sub-problem is how to find additional k_x optimal vertices in T_x with the closest selected

Table 3: The values of $\mathsf{g}_{\mathsf{Yes}}$, g_{No}, and Gain. Here, $\mathcal{T} = \{v_5, v_8\}$.

Node	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
$\mathsf{g}_{\mathsf{Yes}}$	8	1	12	9	10	12	12	12	11
g_{No}	19	1	11	2	5	3	3	3	3
Gain^1	9.85	1	**11.59**	3.4	6.8	4.8	4.8	4.8	4.6
$\mathsf{g}_{\mathsf{Yes}}$	/	0	/	0	1	0	0	0	2
g_{No}	/	1	/	1	3	1	1	1	2
Gain^2	/	0.75	/	0.75	**2.12**	0.75	0.75	0.75	2

vertex w and $\sum k_x \leq k'$. The optimal answer is the best solution among the above two answers.

States and transfer equations. We define two states of $\mathsf{DP}_Y(u, k)$ and $\mathsf{DP}_N(u, w, k)$ first. In the subtree T_u, $\mathsf{DP}_Y(u, k)$ is the minimum value of $\mathsf{f}(\mathcal{S} \cup \{u\}, \mathcal{P} \cap \mathsf{des}(u))$ with selected $(k-1)$-size set $\mathcal{S} \subseteq \mathsf{des}(u)$. Similarly, $\mathsf{DP}_N(u, w, k)$ is the minimum value of $\mathsf{f}(\mathcal{S} \cup \{w\}, \mathcal{P} \cap \mathsf{des}(u))$ with selected k-size set $\mathcal{S} \subseteq \mathsf{des}(u) \setminus \{u\}$ and $w \in \mathsf{anc}(u) \cap \mathcal{Y}$ is the closest selected vertex to u. On the basis of $\mathsf{DP}_Y(u, k)$ and $\mathsf{DP}_N(u, w, k)$, we define the state $\mathsf{DP}(u, w, k)$ as the optimal k-size selection in the subtree T_u with closest selected vertex $w \in \mathsf{anc}(u) \cup \mathcal{P}$, which satisfies the equation as follows.

$$\mathsf{DP}(u, w, k) = \begin{cases} \min\{\mathsf{DP}_Y(u, k), \mathsf{DP}_N(u, w, k)\}, u \in \mathcal{Y}, k \geq 1 \\ \mathsf{DP}_N(u, w, k), \qquad\qquad u \notin \mathcal{Y} \text{ or } k = 0 \end{cases} \tag{5}$$

Next, we propose the transfer equation of $\mathsf{DP}_Y(u, k)$ and $\mathsf{DP}_N(u, w, k)$ as follows.

$$\mathsf{DP}_Y(u, k) = \min\{\sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} \mathsf{DP}(x, u, k_x)\} \\ \text{subject to} \sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} k_x = k - 1. \tag{6}$$

$$\mathsf{DP}_N(u, w, k) = \mathsf{dist}\langle w, u \rangle + \min\{\sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} \mathsf{DP}(x, w, k_x)\} \\ \text{subject to} \sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} k_x = k. \tag{7}$$

Furthermore, we can use the Knapsack dynamic programming technique [26] to tackle the transfer equations in Eqs. 6 and 7. Assume that a number k represents the total capacity. Given a set of vertices $\mathsf{child}(u) \cap \mathcal{P} = \{x_1, ..., x_l\}$, for each vertex x_i where $1 \leq i \leq l$, $\mathsf{DP}(x_i, w, k_{x_i})$ represents an item value and the item volume is $k_{x_i} \leq k$. We assume that $F(i, k')$ is the state that has the minimum value of the first i items with a total of k' capacity. The equation of state transformation is shown as follows.

$$F(i, k') = \min_{0 \leq j \leq k'} \{F(i-1, k'-j) + \mathsf{DP}(x_i, w, j)\}.$$

For initialization, we set $F(i, j) = +\infty$ for $1 \leq i \leq l, 0 \leq j \leq k$ and $F(0, 0) = 0$. The return value is $F(l, k) = \min\{\sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} \mathsf{DP}(x, w, k_x)\}$ with the constraint $\sum_{x \in \mathsf{child}(u) \cap \mathcal{P}} k_x = k$.

Update $\mathsf{g}_{\mathsf{Yes}}(u)$ and $\mathsf{g}_{\mathsf{No}}(u)$. For a vertex u that is questioned, only the state of $\mathsf{anc}(u)$ needs to be recalculated to update $\mathsf{g}_{\mathsf{Yes}}(u)$ and $\mathsf{g}_{\mathsf{No}}(u)$. Algorithm 4 presents the details of $\mathsf{g}_{\mathsf{Yes}}(u)$ and $\mathsf{g}_{\mathsf{No}}(u)$ calculations. First, the algorithm calculates $\mathsf{DP}(u, w, k)$ for all possible states (line 1). Then, for $u \in \mathcal{P} \setminus \mathcal{Y}$, it updates the corresponding states and calculates $\mathsf{g}_{\mathsf{Yes}}(u)$ and $\mathsf{g}_{\mathsf{No}}(u)$ (lines 2-7). For computing $\mathsf{g}_{\mathsf{Yes}}(u)$, it needs to update the states $\mathsf{DP}(v, w, k)$ for all $v \in \mathsf{anc}(u)$, $w \in \mathsf{anc}(v)$ (lines 8-15). To computing $\mathsf{g}_{\mathsf{No}}(u)$, it updates the states $\mathsf{DP}(v, w, k)$ for all $v \in \mathsf{anc}(u)$, $w \in \mathsf{anc}(v) \cap \mathcal{Y}$ (lines 16-26).

Algorithm 5 kBM-DP+: Calculate Expected Gains and Identify q_i

```
Input: $\mathcal{H} = (V, E)$, $\mathsf{p}_{\mathsf{Yes}}(.)$, $\mathsf{p}_{\mathsf{No}}(.)$, $\mathcal{P}$, $\mathcal{Y}$, and $k$.
Output: Question vertex $q_i$.
1: $\mathsf{Gain}_{max} \leftarrow 0$;
2: $\overline{\mathsf{Gain}^i}(v) \leftarrow \mathsf{UBg}_{\mathsf{Yes}}{}^i(v) \cdot \mathsf{p}_{\mathsf{Yes}}(v) + \mathsf{UBg}_{\mathsf{No}}{}^i(v) \cdot \mathsf{p}_{\mathsf{No}}(v)$ for all vertices $v \in \mathcal{P} \setminus \mathcal{Y}$, where $\mathsf{UBg}_{\mathsf{Yes}}{}^i(v) = \mathsf{g}_{\mathsf{Yes}}{}^{(i-1)}(v)$ and $\mathsf{UBg}_{\mathsf{No}}{}^i(v) = \mathsf{g}_{\mathsf{No}}{}^{(i-1)}(v)$.
3: Sort all vertices $v \in \mathcal{P} \setminus \mathcal{Y}$ in the descending order of $\overline{\mathsf{Gain}^i}(v)$;
4: for $v \in \mathcal{P} \setminus \mathcal{Y}$ do
5:     if $\mathsf{Gain}_{max} > \overline{\mathsf{Gain}^i}(v)$ then return $q_i$;
6:     $\mathsf{g}_{\mathsf{Yes}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{calg}_{\mathsf{Yes}}(v, k)$;
7:     $\mathsf{g}_{\mathsf{No}}(v) = \mathsf{g}(\mathcal{Y}, \mathcal{P}, k) - \mathsf{calg}_{\mathsf{No}}(v, k)$;
8:     $\mathsf{Gain}(v) \leftarrow \mathsf{g}_{\mathsf{Yes}}(v) \cdot \mathsf{p}_{\mathsf{Yes}}(v) + \mathsf{g}_{\mathsf{No}}(v) \cdot \mathsf{p}_{\mathsf{No}}(v)$;
9:     if $\mathsf{Gain}_{max} < \mathsf{Gain}(v)$ then
10:        $\mathsf{Gain}_{max} \leftarrow \mathsf{Gain}(v)$; $q_i \leftarrow v$;
11: return $q_i$;
```

Example 4. *Assume that the targets are $\mathcal{T} = \{v_5, v_8\}$, the budget $b = 2$, and $k = 2$ in Figure 3. Table 3 shows the gains of all vertices. In the first round, the algorithm questions v_3 and gets the* Yes *answer. Then, the vertices v_0, v_1 are pruned from $\mathcal{P}$. The vertex v_5 gets the maximum gains in the second round and obtains the* Yes *answer. Note that some leaf vertices get $\mathsf{g}_{\mathsf{Yes}}(v) = 0$ because their parents are better selections even if they get the* Yes *answer. The selections are $\mathcal{S} = \{v_3, v_5\}$ and the penalty is $\mathsf{f}(\mathcal{S}, \mathcal{T}) = 1$.*

Complexity analysis. The calculation of all states takes $O(nhk^2)$ time. The update of each vertex takes $O(h^2dk^2)$ times using Algorithm 4. Overall, the kBM-DP in Algorithm 3 takes $O(bnh^2dk^2)$ time in $O(nhk)$ space for generating b questions.

6.3 Fast algorithms: kBM-Topk and kBM-DP+

In this section, we propose two fast algorithms of kBM-Topk and kBM-DP+. The first method kBM-Topk uses an alternative penalty function to improve the calculation of expected gain. The second method kBM-DP+ develops an upper bound of $\mathsf{Gain}(v)$ to prune unnecessary vertices for updating the expected gains.

6.3.1 kBM-Topk.

The penalty of $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$ is complex to compute, due to the dependence relationship of selections in $\mathcal{S}$. To deal with this issue, we propose a variant penalty function to approximate $\mathsf{g}(\mathcal{Y}, \mathcal{P}, k)$, which can be efficiently computed. We begin with a new definition of selected gain as follows.

$$\mathsf{IG}(x) = \sum_{v \in \mathcal{P} \cap \mathsf{des}(x)} dist\langle r, v \rangle - dist\langle x, v \rangle = dist\langle r, x \rangle \cdot |\mathcal{P} \cap \mathsf{des}(x)|.$$

The selected gain represents the reduced penalty after selecting x. The deeper the selected vertex and the larger the size of descendants, the higher the selected gain. Thus, the general idea of kBM-Topk is to select the top-k vertices that maximize the selected gain. Thus, we propose a new potential penalty function:

$$\mathsf{g}'(\mathcal{Y}, \mathcal{P}, k) = \mathsf{f}(\{r\}, \mathcal{P}) - \max_{\mathcal{S} \subseteq \mathcal{Y}, |\mathcal{S}| \leq k} \sum_{x \in \mathcal{S}} \mathsf{IG}(x). \tag{8}$$

Table 4: The statistics of hierarchical tree datasets.

Name	$\lvert V \rvert$	Depth	Avg Depth	Max Degree	# Queries
Image-COCO	200	5	2.63	37	107,774
ImageNet	74,401	19	8.78	391	16,188,196
Yago3	493,839	17	5.70	44,538	4,440,378

Compared with kBM-DP, the kBM-Topk algorithm uses the same framework but adopts a heuristic penalty function $\mathrm{g}'(\mathcal{Y}, \mathcal{P}, k)$. The kBM-Topk algorithm takes $O(bnh \log n)$ time using $O(n)$ space for generating b questions. The detailed kBM-Topk algorithm and complexity analysis are available in [35].

6.3.2 kBM-DP+. In this section, we propose a pruning optimization to accelerate the algorithm kBM-DP. The general idea is to design an upper bound of expected gain and skip the update of $\mathrm{Gain}(v)$ for those vertices that are disqualified for achieving the largest $\mathrm{Gain}(v)$ in $\mathcal{P} \setminus \mathcal{Y}$. In this way, we can prune lots of vertices in most cases at each round of question asking, and quickly identify a vertex q_i with the largest gain.

An upper bound of $\mathrm{Gain}(v)$. Consider a vertex v at the i-th round of question asking, the expected gain is denoted as $\mathrm{Gain}^i(v)$. Then, we have an upper bound of $\mathrm{Gain}^i(v)$, denoted as $\overline{\mathrm{Gain}^i}(v)$, satisfying $\overline{\mathrm{Gain}^i}(v) = \mathrm{UBg_{Yes}}^i(v) \cdot \mathrm{p_{Yes}}(v) + \mathrm{UBg_{No}}^i(v) \cdot \mathrm{p_{No}}(v)$, where $\mathrm{UBg_{Yes}}^i(v) = \mathrm{g_{Yes}}^{(i-1)}(v)$ and $\mathrm{UBg_{No}}^i(v) = \mathrm{g_{No}}^{(i-1)}(v)$. Note that if v is pruned in the previous round, we will set $\mathrm{UBg_{Yes}}^i(v) = \mathrm{UBg_{Yes}}^{(i-1)}(v)$ and $\mathrm{UBg_{No}}^i(v) = \mathrm{UBg_{No}}^{(i-1)}(v)$. We observe that both the Yes gain and No gain decrease with more questions asked in most cases, due to the decreased $\mathcal{P}$ and increased $\mathcal{Y}$. Thus, we have $\mathrm{UBg_{Yes}}^i(v) \geq \mathrm{g_{Yes}}^i(v)$ and $\mathrm{UBg_{No}}^i(v) \geq \mathrm{g_{No}}^i(v)$. As a result, $\overline{\mathrm{Gain}^i}(v) \geq \mathrm{Gain}^i(v)$.

Algorithm. kBM-DP+ is a variant approach of kBM-DP in Algorithm 3 using the pruning optimization in Algorithm 5, which calculates expected gains and identifies the vertex q_i for question asking (replacing lines 7-8 of Algorithm 3). Specifically, the algorithm first computes all upper bounds for vertices $v \in \mathcal{P} \setminus \mathcal{Y}$ and then sorts the vertices in descending order of upper bounds (lines 2-3). Next, it calculates the expected gain $\mathrm{Gain}(v)$ and prunes disqualified vertices with an upper bound $\overline{\mathrm{Gain}^i}(v) < \mathrm{Gain}_{max}$ where Gain_{max} keeps updated with the largest value of all possible expected gains (lines 4-10). Finally, it returns a vertex q_i with the largest expected gain. Note that we offline pre-compute the $\mathrm{g_{Yes}}$ and $\mathrm{g_{No}}$ of all vertices for the first question in $\mathcal{H}$, which asks the same question for any targets.

7 EXPERIMENTS

Datasets. We use three real datasets of hierarchical trees, whose detailed statistics are summarized in Table 4. First, *ImageNet* [3, 10] is a hierarchical image dataset based on WordNet. It has 74,401 taxonomy vertices and 16 million images with ground-truth labels. Second, we generate a small hierarchy with 200 taxonomy vertices from COCO [18] and ImageNet [10], denoted as *Image-COCO*, ensuring the successful and efficient running of all tested algorithms. For target search on Image-COCO and ImageNet, we randomly select a set of 1,000 images with a single label and another set of 1,000 images with multiple labels for the SingleTarget and MultipleTargets problems, respectively. Third, *Yago3* [4, 20] is a knowledge base from multilingual Wikipedias. We use the ontology structure yagoTaxonomy as the hierarchy for testing. It contains 493,839 taxonomy vertices, where an edge $\langle v, u \rangle$ means vertex u is a "subClassOf" vertex v. Moreover, Yago3 contains 4,440,378 objects from yagoTypes, where each object may have a single label or multiple labels. For both the SingleTarget and MultipleTargets problems, we select two sets of objects with a single label and with multiple labels, respectively, using two methods. In the first method, we randomly select 1,000 labeled objects from Yago3, denoted as *Yago3-I*. In the second method, we first randomly select 1,000 categories from Yago3 and then pick a random labeled object under each selected category, denoted as *Yago3-II*.

Comparison methods. We compare our algorithms with state-of-the-art methods HGS [23], IGS [29], and BinG [17]. Specifically,

- HGS: a dynamic programming Human-GS method for identifying multiple targets with a bounded number of questions, which generates b questions offline in a non-interactive setting [23]. Following the algorithms in [23], we implement two methods, *Single-Bounded* and *Multi-Bounded*, for identifying a single target and multiple targets, respectively.
- IGS: an interactive graph search algorithm for identifying a single target [29]. The algorithm decomposes a hierarchy into connected paths and finds the target through a series of binary searches on individual paths.
- BinG: a greedy algorithm for identifying a single target, which asks questions using an optimal vertex that prunes the largest number of vertices [17]. It prunes the vertices $\mathcal{P} \setminus \mathrm{des}(u)$ for $\mathrm{reach}(u) = \mathrm{Yes}$. To identify multiple targets, we implement a variant BinG method, which only prunes $\mathcal{P} \cap \mathrm{anc}(u) \setminus \{u\}$ for $\mathrm{reach}(u) = \mathrm{Yes}$.

Note that both IGS and BinG can ask unlimited questions to identify the targets. In our problem setting, we terminate the algorithms of IGS and BinG after asking b questions. We also evaluate and compare our proposed algorithms as follows.

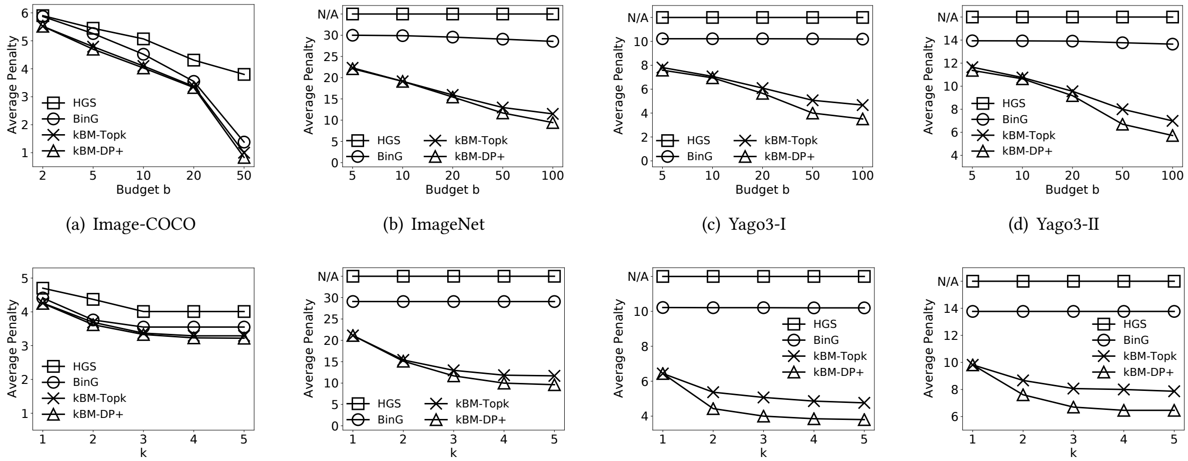
- STBIS: identifies a single target in Algorithm 2.
- kBM-DP: a dynamic programming based method for identifying multiple targets in Algorithms 3 and 4.
- kBM-Topk: uses an independent penalty function to select top-k vertices in Section 6.3.1.
- kBM-DP+: uses an upper bound pruning technique to accelerate kBM-DP in Algorithm 5.

After asking b questions, all algorithms return the selections from Yes-candidates in the same way following our kBM-IGS framework.

Evaluation metrics and parameter settings. For quality evaluation, we use the penalty $\mathrm{f}(\mathcal{S}, \mathcal{T})$ to measure the closeness between selections $\mathcal{S}$ and targets $\mathcal{T}$ by Def. 3. For each experiment, we report the averaged penalty score of searching targets on 1,000 selected images/objects. By default, we set the budget $b = 50$, and assign $k = 1$ and $k = 3$ respectively for the SingleTarget and MultipleTargets problems. The initial probability of each vertex v is set as $\mathrm{pr}(v) = \frac{k}{n}$. We denote the running time as *INF* and the penalty result as *N/A*, if an algorithm cannot finish within 100 hours.

Table 5: Quality evaluation (penalty scores) of different methods for identifying a single target.

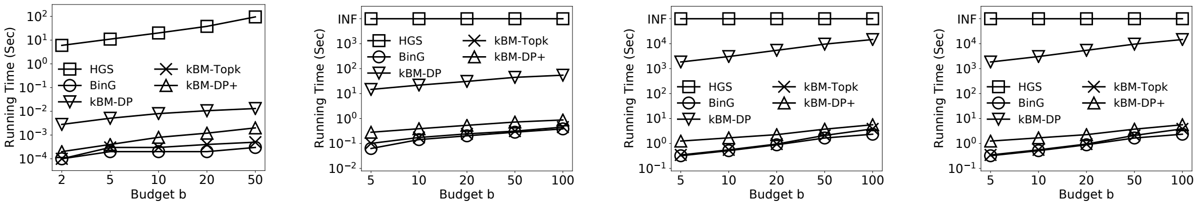
	Image-COCO				ImageNet				Yago3-I				Yago3-II			
Budget b	HGS	IGS	BinG	STBIS	HGS	IGS	BinG	STBIS	HGS	IGS	BinG	STBIS	HGS	IGS	BinG	STBIS
5	1.48	1.42	1.21	**1.20**	4.59	4.50	3.98	**3.89**	2.04	2.03	**1.60**	1.61	2.61	2.60	2.30	**2.27**
10	1.34	1.15	0.79	**0.78**	4.50	3.87	3.10	**3.04**	1.92	1.63	1.15	**1.05**	2.48	2.20	1.69	**1.53**
20	1.20	0.77	**0.47**	**0.47**	4.34	3.02	1.80	**1.79**	1.83	1.18	0.71	**0.65**	2.30	1.61	1.12	**1.05**
50	0.98	0.41	**0.18**	**0.18**	4.21	1.41	**0.67**	**0.67**	1.71	0.82	0.41	**0.39**	2.14	1.04	0.73	**0.67**
100	0.52	0.19	**0.00**	**0.00**	3.79	0.68	**0.28**	**0.28**	1.66	0.71	0.30	**0.27**	2.07	0.90	0.56	**0.54**



(a) Image-COCO (b) ImageNet (c) Yago3-I (d) Yago3-II

(e) Image-COCO (f) ImageNet (g) Yago3-I (h) Yago3-II

Figure 5: Quality evaluation (penalty scores) of different methods for identifying multiple targets.



(a) Image-COCO (b) ImageNet (c) Yago3-I (d) Yago3-II

Figure 6: The running time of the five algorithms that ask b questions on all datasets.

EXP-1: Quality evaluation of the SingleTarget **problem.** Table 5 shows the penalty results of four methods HGS, IGS, BinG, and STBIS for identifying a single target. For each dataset, we test five different budgets of b, varying from 5 to 100. The smaller the penalty scores, the closer the selections to the hidden targets. All methods get lower penalty scores with increased budget b as more questions are asked to obtain better selections. Our method STBIS achieves the best performance in all tests, except for one case of $b = 5$ on Yago3-I. In particular, it outperforms HGS by 23%–1,253%. While BinG has a competitive performance with STBIS, it is much worse than our methods in the more challenging MultipleTargets problem as will be shown in EXP-2.

EXP-2: Quality evaluation of the MultipleTargets **problem.** We evaluate four methods HGS, BinG, kBM-Topk, and kBM-DP+ for identifying multiple targets. Figures 5(a)-5(d) and Figures 5(e)-5(h) report the penalty results on all datasets by varying budget b and selection size k, respectively. Several observations are made. First, HGS has the largest penalty scores on the small dataset Image-COCO as shown in Figures 5(a) and 5(e). On the three large datasets in Figures 5(b)-5(d) and 5(f)-5(h), HGS fails to finish within 100 hours, due to its high time complexity. Second, compared with BinG, our methods kBM-DP+ and kBM-Topk get smaller penalty scores by achieving an average of 2.1x better results. The main reason is that BinG tends to ask questions on the vertices at the bottom levels, which is likely to get a No answer with little gain of

Question	Label	reach(q_i)	depth(q_i)	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{Y}\rvert$	f($\mathcal{S}^*, \mathcal{T}$)
q_0	**animal**	Yes	0	3,998	1	11
q_1	**vertebrate**	Yes	2	3,996	3	7
q_2	**mammal**	Yes	3	3,995	4	6
q_3	invertebrate	No	1	3,219	4	6
q_4	**aquatic vertebrate**	Yes	3	3,219	5	5
q_5	**fish**	Yes	4	3,218	6	4
q_6	bird	No	3	2,347	6	4
q_7	bony fish	No	5	1,812	6	4
q_8	**carnivore**	Yes	5	1,810	8	2
q_9	dog	No	7	1,587	8	2
q_{10}	reptile	No	3	1,291	8	2
q_{11}	ungulate	No	5	988	8	2
q_{12}	**felid (cat family)**	Yes	6	987	9	1

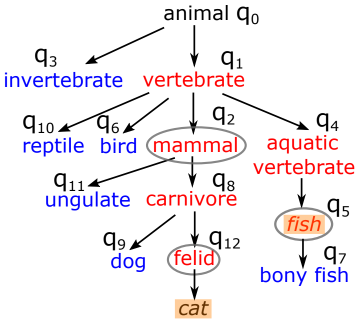
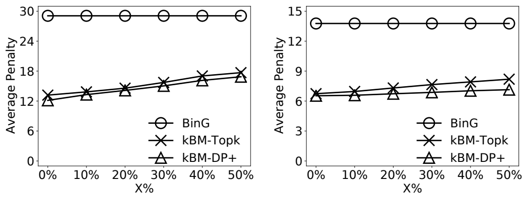


Figure 7: Case study on the "animal" hierarchy in ImageNet. The targets $\mathcal{T}$ = {"fish", "cat"}, $b = 12$, and $k = 3$. The selection set of our algorithm is $\mathcal{S}$ ={"fish", "mammal", "felid"}, in which "felid" is the parent of "cat". The penalty score is $f(\mathcal{S}, \mathcal{T}) = 1$.

reducing target penalties. In contrast, our methods kBM-Topk and kBM-DP+ aim at asking questions on the vertices with the largest expected gains based on the potential target distribution, thereby achieving a better performance. Moreover, with increased budget b and target number k, kBM-Topk and kBM-DP+ get an even better performance with lower penalty scores. Finally, between kBM-DP+ and kBM-Topk, kBM-DP+ incurs less penalties because it has a better gain function for identifying diverse selections.

EXP-3: Efficiency evaluation. Figure 6 shows the running time results of different algorithms for identifying multiple targets. All methods take more time with increased budget b. Among all algorithms, HGS and kBM-DP are the most inefficient. In particular, HGS fails to finish on ImageNet, Yago3-I, and Yago3-II within 100 hours. On the other hand, kBM-Topk and BinG are the fastest algorithms by adopting simple penalty functions to generate questions. Specifically, kBM-Topk runs 2.8x faster on average than kBM-DP+ for different parameters of b in Figures 6(a)-(d).

EXP-4: Quality evaluation with wrong answers. We conduct a quality evaluation of our methods where human mistakes are not eliminated and the workers give wrong answers. For each dataset, we randomly select $X\%$ objects out of 1,000 objects and treat them as difficult objects. We vary $X \in [0, 50]$ on the ImageNet and Yago3-I datasets. For each question that involves a difficult object, the workers have a probability of giving a wrong answer, denoted as p. In the experiment, we set *the wrong probability* $p = 10\%$ and *budget* $b = 50$. Figure 8 shows the penalty results when varying the percentage of difficult objects. The quality performances of kBM-Topk and kBM-DP+ are only slightly degraded with the increasing percentage of difficult objects, demonstrating their resilience to wrong answers. Moreover, our methods kBM-Topk and kBM-DP+ still win BinG by at least 40%, even with wrong answers.



(a) ImageNet

(b) Yago-I

Figure 8: Quality evaluation with wrong answers.

EXP-5: Case study of image categorization. We conduct a case study of interactive search to identify multiple targets on ImageNet. We extract the "animal" sub-hierarchy of ImageNet, which contains nearly 4,000 labels. We use the image shown in Figure 1(b) with $\mathcal{T}$ ={"fish", "cat"} for search. The left table in Figure 7 shows the detailed process and statistics of all interactive questions asked by kBM-DP+ with $b = 12$ and $k = 3$. For each question vertex q_i, we report the label of q_i, the answer reach(q_i), the depth of q_i in $\mathcal{H}$, $|\mathcal{P}|$, $|\mathcal{Y}|$, and the penalty $f(\mathcal{S}^*, \mathcal{T})$. We also show the questioned vertices in a simplified hierarchy on the right side of Figure 7. The red vertices get a Yes answer and the blue vertices get a No answer. The interactive process clearly shows that our questions approach the targets quickly in a top-down manner within 12 questions, which achieves a very small penalty of 1 between selections and targets. Finally, kBM-DP+ identifies the selections $\mathcal{S}$ = {"fish", "mammal", "felid"}. Note that "felid" means the cat family, which is the parent of the target "cat".

8 CONCLUSION

In this paper, we study the problem of kBM-IGS to identify multiple targets in a hierarchy using a constrained budget of interactive questions. To effectively tackle the problem, we propose a novel kBM-IGS framework to select the vertex with the maximum expected gain to ask question. On the basis of the kBM-IGS framework, we develop STBIS algorithm to identify a single target and a dynamic programming based method kBM-DP to identify multiple targets. To further improve the efficiency, we propose two heuristic algorithms kBM-Topk and kBM-DP+ to ask question on the vertex with the best alternative gain. Extensive experiments validate the effectiveness and efficiency of our proposed algorithms.

ACKNOWLEDGMENTS

This paper is supported by HK RGC Grants Nos. 22200320, 12201119, 12200819, 12201518, HK RGC CRF C6030-18G, NSFC No. 62072138, Open Research Projects of Zhejiang Lab (No. 2021KC0AB02), and Guangdong Basic and Applied Basic Research Foundation (No. 2019B1515130001). Xin Huang is the corresponding author.

REFERENCES

[1] http://mturk.com.

[2] http://crowdflower.com.

[3] http://image-net.org/download-API.

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

[5] J. Barr and L. F. Cabrera. Ai gets a brain. *Queue*, 4(4):24–29, 2006.

[6] J. Bragg, D. S. Weld, et al. Crowdsourcing multi-label classification for taxonomy creation. In *HCOMP*, pages 25–33, 2013.

[7] K. Chakrabarti, S. Chaudhuri, and S.-w. Hwang. Automatic categorization of query results. In *SIGMOD*, pages 755–766, 2004.

[8] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay. Cascade: Crowdsourcing taxonomy creation. In *SIGCHI*, pages 1999–2008, 2013.

[9] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Computing Surveys*, 51(1):1–40, 2018.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[11] Y. Gao, X. Han, X. Wang, W. Huang, and M. Scott. Channel interaction networks for fine-grained image categorization. In *AAAI*, pages 10818–10825, 2020.

[12] L. Guo, G. Fan, and W. Sheng. Dual graphical models for relational modeling of indoor object categories. In *CVPR Workshops*, pages 1007–1013, 2019.

[13] A. M. E. W. D. Karger and S. M. R. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[14] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NeurIPS*, pages 1953–1961, 2011.

[15] L. Karlinsky, J. Shtok, Y. Tzur, and A. Tzadok. Fine-grained recognition of thousands of object categories with single-example training. In *CVPR*, pages 4113–4122, 2017.

[16] K. Li and G. Li. Approximate query processing: What is new and where to go? *Data Science and Engineering*, 3(4):379–397, 2018.

[17] Y. Li, X. Wu, Y. Jin, J. Li, and G. Li. Efficient algorithms for crowd-aided categorization. *PVLDB*, 13(8):1221–1233, 2020.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[19] Z. Liu, S. Hu, Y. Yin, J. Chen, K. Chiew, L. Zhang, and Z. Wu. Interactive rare-category-of-interest mining from large datasets. In *AAAI*, pages 4965–4972, 2020.

[20] F. Mahdisoltani, J. Biega, and F. M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2015.

[21] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[22] J. Ni, J. Li, and J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*, pages 188–197, 2019.

[23] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: It is okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[24] A. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom. Datasift: a crowd-powered search toolkit. In *SIGMOD*, pages 885–888, 2014.

[25] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.

[26] D. Pisinger. Algorithms for knapsack problems. 1995.

[27] Y. Sun, A. Singla, D. Fox, and A. Krause. Building hierarchies of concepts via crowdsourcing. In *IJCAI*, pages 844–853, 2015.

[28] P. Tang, M. Jiang, B. N. Xia, J. W. Pitera, J. Welser, and N. V. Chawla. Multi-label patent categorization with non-local attention-based graph convolutional network. In *AAAI*, pages 9024–9031, 2020.

[29] Y. Tao, Y. Li, and G. Li. Interactive graph search. In *SIGMOD*, pages 1393–1410, 2019.

[30] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[31] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[32] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.

[33] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[34] D. Zhou, L. Chen, and Y. He. An unsupervised framework of exploring events on twitter: Filtering, extraction and categorization. In *AAAI*, pages 2468–2475, 2015.

[35] X. Zhu, X. Huang, B. Choi, J. Jiang, Z. Zou, and J. Xu. Budget constrained interactive search for multiple targets. *arXiv preprint arXiv:2012.01945*, 2020.