# Efficient Example-Guided Interactive Graph Search

Zhuowei Zhao
*University of Melbourne*
zhuoweiz1@student.unimelb.edu.au

Junhao Gan
*University of Melbourne*
junhao.gan@unimelb.edu.au

Jianzhong Qi
*University of Melbourne*
jianzhong.qi@unimelb.edu.au

Zhifeng Bao
*RMIT University*
zhifeng.bao@rmit.edu.au

***Abstract*—We study the problem of *interactive graph search* (IGS). Given a query entity $\varphi$, the goal is to identify the target concept in a *directed acyclic graph* (DAG) concept hierarchy $H$, which best describes $\varphi$, through interactions with an oracle. In each interaction, a question in the form of "Does $\varphi$ belong to concept $u$?" is asked and the oracle can only answer either `YES` or `NO`. The efficiency of an IGS algorithm is measured by the number of questions asked, to identify the target concept, which is referred to as *query cost*.**

**In theory aspect, we propose the *Target-Sensitive IGS* (*TS-IGS*) algorithm that achieves a query cost complexity of $O(\log n \cdot \log \frac{L}{\log n} + d \cdot \log_d n)$, where $L$ is the length of the path from the root of $H$ to the target concept. When $L \in O(\log n)$, our *TS-IGS* matches the known lower bound [1].**

**In practice aspect, we propose an algorithm called *Example-Guided IGS* (*EG-IGS*) that exploits the knowledge of entities and asks *promising questions* guided by *examples* similar to $\varphi$. We prove that *EG-IGS* achieves a finer-grained query cost bound than that of *TS-IGS*, and is extremely efficient in practice.**

**Extensive experiments on six real-world datasets (including images, texts, and gene sequences) show that our *EG-IGS* outperforms all the existing competitors by up to two orders of magnitude in terms of query cost, and is robust in various settings. To further demonstrate the real feasibility of our *EG-IGS* technique, we develop a fully-automatic Amazon product categorization demo system with GPT-3.5 serving as the oracle.**

## I. Introduction

High-quality labelled data is of great importance to various data-driven applications and supervised learning tasks, e.g., image classification [2]–[4], entity resolution [5]–[7], corpus annotation [8], [9], sentiment analysis [10], etc. In these tasks, the labelled data not only serves as reliable training data, but also often serves as ground truth for model tuning. Labelling data is notoriously challenging, due to the large volume of the datasets, the ambiguity and subjectivity of the tasks, and the expensive (hiring/training) cost of the annotators.

**Data Labelling Task in a Concept Hierarchy.** In this paper, we focus on the task of labelling data with labels in a *concept hierarchy* $H$. A concept hierarchy $H$ is a *directed acyclic graph* (DAG), where each node represents a *concept* and a directed edge from a node $u$ to another node $v$ means "concept $v$ is a *kind of* concept $u$". For example, in Figure 1, "`animal`" is a kind of "`entity`" and "`mammal`" is a kind of "`animal`". Given a *query entity* $\varphi$ (e.g., an image to be classified), the data labelling task aims to find the target concept $t^*_\varphi$ in the concept hierarchy $H$ that *best describes* $\varphi$ (i.e., $t^*_\varphi$ is the *deepest* node in $H$ to which $\varphi$ belongs). Consider the example shown in Figure 1; given the image of a cat $\varphi$, while $\varphi$ is a kind of

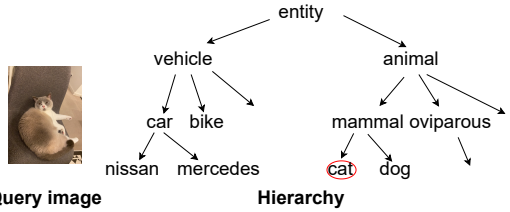



Fig. 1: An Image Labelling Task Example

`animal`, the concept that best describes $\varphi$ is `cat`, and hence, $t^*_\varphi = $ `cat`.

**The Interactive Graph Search (IGS) Problem.** [1] formulate the above data labelling task in a concept hierarchy as the *Interactive Graph Search* (IGS) problem, where the data labelling task is performed by asking an *oracle* questions. Each *question* to the oracle is in the form of "*Is the given entity $\varphi$ a kind of concept $t$?*", and the response from the oracle to each question can be either `YES` or `NO` *only*. For example, in Figure 1, one may ask a question like "Is $\varphi$ a kind of `animal`?", and the answer from the oracle will be "`YES`". Given a query entity $\varphi$, if the oracle returns `YES` to the question with respect to $\varphi$ and a concept $t$, then $t$ is a *`YES`-concept* for $\varphi$; otherwise, $t$ is a *`NO`-concept* for $\varphi$. A node $t^*_\varphi$ is confirmed to be the *target concept* for $\varphi$ if $t^*_\varphi$ satisfies the following two conditions:

1) $t^*_\varphi$ is a `YES`-concept for $\varphi$,
2) either $t^*_\varphi$ is a *leaf node* (i.e., having no out-neighbours in $H$), or all the out-neighbours of $t^*_\varphi$ in $H$ are `NO`-concepts for $\varphi$.

To this end, the goal of the IGS problem is to identify the target concept $t^*_\varphi \in H$ for the query entity $\varphi$ with the oracle. The efficiency of an algorithm is measured by the total number of questions asked to the oracle, which we refer to as the *query cost* of the algorithm.

**Applications.** The IGS problem captures a wide range of data labelling tasks in practice. Below are three typical application scenarios, which require high, medium, and light levels of human involvement efforts, respectively.

*Scientific Data Labelling.* This scenario often requires an oracle of professionals with domain expertise to provide high-accuracy answers (e.g., classifying viruses based on gene sequences). Each question to the oracle can be expensive in terms of monetary cost.

*Crowd-sourcing Data Labelling.* In this scenario, the crowd-sourcing platform serves as the oracle and typical tasks include

image classification and product categorization. While the monetary cost of a question might not be as expensive as the previous one, the *turn-around time* can be long to receive answers from crowd workers.

*Fully-automatic Data Labelling.* In this scenario, an AI program often serves as the oracle, such as our demo system to be introduced in Section VII. Although the monetary cost and turn-around time are relatively low, the correctness of the answers to the questions might suffer from certain failure probability. As a result, the more questions asked, the higher chance to fail in the data labelling task.

From the above scenarios, one can see that reducing the *actual query cost* (not just the theoretical complexity) to identify the target concepts is crucial to the *monetary cost*, the *turnaround time* as well as *the result quality of the labelling tasks*. Moreover, it is worth mentioning that, unlike the running time of algorithms whose absolute difference might be small (in milliseconds), the actual query cost is highly *constant-sensitive*: if the query cost is reduced by half, one will save half of the monetary cost immediately.

**Existing Algorithms and Their Limitations.**

*A Brute-force Algorithm.* A straightforward brute-force algorithm for solving the IGS problem is to "descend" a path from the *root* to $t^*_\varphi$ step by step. At each step at a YES-concept $u$ (initially, $u$ is the root), the brute-force algorithm asks a question for each out-neighbour of $u$ until the first YES-concept $v$ is found or all the out-neighbours are NO-concepts (in this case, $u$ is returned as $t^*_\varphi$). The algorithm updates $u$ to be $v$ and repeats until $t^*_\varphi$ is identified. Clearly, the query cost complexity of this algorithm is bounded by $O(d \cdot h)$, where $d$ is the maximum out-degrees of the nodes in $H$ and $h$ is the length of the longest path in $H$.

*The State-of-the-Art Algorithm.* [1] proposed an algorithm called *DFS-Interleave*. The basic idea of the *DFS-Interleave* algorithm is to interleave *binary search* and the above "brute-force" out-neighbourhood checking. Specifically, it first computes a *special DFS-tree* $T$ of $H$. With such a tree $T$, the path from the root to $t^*_\varphi$ can *conceptually* be decomposed into a sequence of paths, where every two consecutive paths are connected with a tree edge in $T$. Starting from the first path in the sequence, *DFS-Interleave* performs a binary search on the current path to find the deepest YES-concept $u$ on this path and then invokes the "brute-force" out-neighbourhood checking for $u$ to "jump" to the next path in the sequence. This interleaving process is repeated until $t^*_\varphi$ is found. Tao et al. showed that the query cost complexity of their *DFS-Interleave* algorithm is bounded by $O(\log h \cdot \log n + d \cdot \log_d n)$, and the worst-case query cost complexity lower bound for solving the IGS problem is $\Omega(d \cdot \log_d n)$.

*Limitations.* Despite the theoretical bound, the actual query cost of *DFS-Interleave* may suffer in practice, especially when $H$ is *short* and *flat*, in the sense that, a concept may have many sub-concepts but the levels of concepts might not be very deep. To see this, on the YAGO3 dataset (in Section VI) with a concept hierarchy of $569,104$ nodes with maximum out-degree $d = 44,983$ and $h = 20$, the *DFS-Interleave* algorithm requires, on average over $1,000$ queries, to ask $2,484$ questions per query. If this task were conducted on the Amazon Mechanical Turk platform [11] with the lowest pay rate ($\$0.01$ per question), it would have taken $\$24.84$ on average just to label a single entity, not to mention the potentially huge turnaround time of receiving answers to these $2,484$ questions by crowd-sourcing! Thus, there still remains a big gap between theory and practice.

**Our Contributions.** We summarize our contributions as follows.

*An Improved IGS Algorithm* (Section III). In the theory aspect, we propose a new algorithm, called *Target-Sensitive IGS* (*TS-IGS*) algorithm, whose query cost is *sensitive* to the location of $t^*_\varphi$ in the aforementioned special DFS-tree $T$. Specifically, the query cost of our *TS-IGS* algorithm is bounded by $O(\log n \cdot \log \frac{L}{\log n} + d \cdot \log_d n)$, where $L$ is the length of the path from the root of $T$ to $t^*_\varphi$, and $L \leq h$ always holds. In particular, when $L \in O(\log n)$, we show that this query cost bound becomes $O(d \cdot \log_d n)$ which is *optimal* up to a constant factor; and when $L \in O(\text{polylog } n)$, this bound becomes $O(\log n \cdot \log\log n + d \cdot \log_d n)$ which is strictly better than that of *DFS-Interleave*, given that $h$ can be as large as $n$ in the worst case.

*Our Example-Guided IGS Algorithm* (Section IV). In practicality aspect, we propose a novel algorithm, called *Example-Guided IGS* (for short, *EG-IGS*). As its name suggests, in addition to the knowledge of the hierarchy $H$, our *EG-IGS* algorithm further explores the knowledge of another important aspect of the IGS problem – the query entities! The basic idea of *EG-IGS* is to ask *promising questions* guided by *examples* of the concepts in the hierarchy $H$, where an *example* of a concept $t$ is an entity for which $t$ is the target concept. Given a query entity $\varphi$, *EG-IGS* works on $T$ as follows:

Step 1. Collect a set $\mathcal{E}$ of examples *similar* to $\varphi$;
Step 2. Derive a set $Q$ of promising questions (or equivalently, concepts) from the paths from the root to the corresponding concepts of the examples in $\mathcal{E}$;
Step 3. Identify the *deepest* YES-concept $t$ in $Q$;
Step 4. Invoke our *TS-IGS* algorithm on the sub-tree of $T$ rooted at $t$ to identify the target concept.

As we will discuss in detail, the first two steps are *query-cost-free*, while the last two steps need to ask questions to the oracle. We show that the query cost of the last two steps is bounded by $O(\log n \cdot \log \frac{L'}{\log n} + d \cdot \log_d n)$, where $L'$ is the length of the path in the tree $T$ from the YES-concept $t$ (identified from the promising questions in Step 3) to $t^*_\varphi$ and $L' \leq L$ always holds.

Our *EG-IGS* algorithm not only improves the state-of-the-art query cost complexity but also runs *extremely* fast in practice. As we shall see in Section VI, on the aforementioned YAGO3 dataset, the average cost of *EG-IGS* is just 13 per query (whereas the number for *DFS-Interleave* is $2,484$), a *two-orders-of-magnitude* improvement.

The key to the success of our *EG-IGS* algorithm is based on the following two crucial observations:

- Entities belonging to the same target concept are often similar.
- The target concepts of similar entities should not be too far away in $T$ from each other.

Therefore, it is often the case that either $t^*_\varphi$ is captured in the promising question set $Q$, or the deepest YES-concept $t$ identified from $Q$ in Step 3 is not far from $t^*_\varphi$. As a result, $L'$ is often small. In our experiments, $L' \leq 14 \in O(\log n)$ holds across all the six real datasets in all settings. In this sense, *EG-IGS* is indeed optimal in practice.

*Example Acquisition* (Section IV-C). One "missing piece" in the above description of our *EG-IGS* is how to obtain examples for the concepts. Perhaps, our solution to address this is one of the most interesting parts in the *EG-IGS* algorithm, which stems from the following observation:

> *While identifying a proper label for a given entity might be challenging, finding entities with respect to a given label can be much simpler.*

For instance, identifying a cat from a given image might be challenging, but finding an image of cat can be as simple as searching the keyword cat with Google Images. Indeed, the examples for the datasets in our experiments are all obtained from external sources such as Google Images, Wikipedia, and Amazon websites. It is worth mentioning that obtaining examples for the concepts is completely *query-cost-free*, i.e., no questions to the oracle are needed. Moreover, even when the external example sources are unavailable, our *EG-IGS* still can accumulate examples by recording the query results. In our experiments, after about 200 examples accumulated from *cold start*, the query cost of *EG-IGS* reduces substantially. Therefore, the practicality of our *EG-IGS* algorithm is obvious.

*Extensive Experiments* (Section VI). We conducted extensive experiments on six real datasets involving different types of entities (images, texts, and gene sequences), whose labelling tasks exhibit different levels of human involvement efforts. The experimental results show that our *EG-IGS* algorithm outperforms all the four existing competitors by up to two orders of magnitude (over 180 times). When the external example sources are unavailable, our algorithm still outperforms all the competitors by up to 2-3 times. As mentioned earlier, this query cost improvement will immediately reflect in the monetary cost saved in crowd-sourcing tasks.

*A Data Labelling Demo System with GPT-3.5* (Section VII). Last but not the least, to further demonstrate the practicality and feasibility of our *EG-IGS* algorithm, we developed a fully-automatic system for product categorization for the Amazon website, where GPT-3.5 [12] is adopted as the oracle in our algorithm to eliminate the need for human effort. Our demo system sheds light on the feasibility of building fully-automatic data labelling systems with a *Generative Pre-trained Transformer* (GPT), which we believe will be of independent interest to the field.

## II. Preliminaries

### A. Problem Definition

**Concept Hierarchy.** A *concept hierarchy* is defined as a *directed acyclic graph* (DAG), denoted by $H = \langle V, E \rangle$, where $V$ is a set of *concepts* (a.k.a. *nodes*), and $E$ is a set of directed edges. An edge $(u, v) \in E$ represents the fact that concept $v$ is a *sub-concept* of, e.g., "a kind of", concept $u$, and $(u, v)$ is called an *in-coming edge* of $v$, and $v$ is called an *out-neighbour* of $u$. For a concept $u \in V$, the number of its out-neighbours is defined as $u$'s *out-degree* and denoted by $d_u$. For example, in Figure 1, the edge $(\texttt{mammal}, \texttt{cat})$ indicates that concept cat is a sub-concept of, i.e., a kind of, concept mammal. This sub-concept relation is *transitive*: if there exists a directed path from a concept $u$ to a concept $w$ in $H$, then $w$ is a sub-concept of $u$.

Without loss of generality, $H$ is deemed as *single rooted*, i.e., there is only one concept in $H$ with no incoming edge. Otherwise, we can add a virtual concept $u$ (with no incoming edge) to $H$ as the only root and add edges from $u$ to all the nodes in $H$.

**Entity and Its Target Concept.** An *entity* $\varphi$ is a *concrete object* which belongs to all the concepts in $H$ on the path(s) from the *root* to a *unique target concept*, [1] denoted by $t^*_\varphi$, which is the *deepest* concept (i.e., the concept farthest from the root) in $H$. Since $H$ is a DAG, there can be multiple paths from the root to $t^*_\varphi$. An entity can take various forms such as an image, a product on an e-commerce site, a text document, and a gene sequence. For example, in Figure 1, an image of a cat $\varphi$ is an entity, and it belongs to all the concepts on the path: $\texttt{entity} \rightarrow \texttt{animal} \rightarrow \texttt{mammal} \rightarrow \texttt{cat}$ in the concept hierarchy $H$. And cat is the target concept $t^*_\varphi$ for the image $\varphi$.

**Questions and the Oracle.** Given an entity $\varphi$, the process to identify the target concept $t^*_\varphi$ is only through the *interactions* with an *oracle*. Each interaction with the oracle is a question-and-answer process, that is, given an entity $\varphi$ and a concept $u \in V$, a *question* denoted by $q(\varphi, u)$, asks the *oracle*:

> *Does entity $\varphi$ belong to concept $u$, or equivalently, is $u$ reachable to the target concept $t^*_\varphi$ in $H$?*

The answer from the oracle to a question $q(\varphi, u)$ can be either YES or NO only, denoted by $q(\varphi, u) = 1$ and $q(\varphi, u) = 0$, respectively. If $q(\varphi, u) = 1$, then $u$ is a *YES-concept* for $\varphi$; otherwise, $u$ is a *NO-concept* for $\varphi$. Moreover, for a YES-concept $u$, all the nodes on the path from the root to $u$ in $H$ are all YES-concepts. On the other hand, for a NO-concept $u$, *none* of the nodes that are reachable from $u$ in $H$ would be a YES-concept. In order to decide whether a concept $u \in V$ is indeed the target concept $t^*_\varphi$, one has to verify two conditions:

1) $u$ is a YES-concept for $\varphi$, and
2) either $u$ is a *leaf node* (i.e., having no out-neighbours in $H$), or all the out-neighbours of $u$ are NO-concepts for $\varphi$.

[1] Following [1] we consider that the target concept of an entity is unique.

**Definition 1** (Interactive Graph Search [1])**.** *Consider a concept hierarchy $H$ and an oracle; given a query entity $\varphi$, the goal of the* Interactive Graph Search *(IGS) problem is to identify the target concept $t^*_\varphi$ via the interactions (by asking questions) with the oracle.*

**Algorithm Efficiency Measurement.** The *efficiency* of an algorithm solving the IGS problem is measured by the total number of questions asked during the interactions with the oracle to identify the target concept for any query entity $\varphi$. This number of questions is called the *query cost* of the algorithm.

### *B. Existing Algorithms and Their Limitations*

Here, we introduce some details of a straightforward baseline algorithm and the state-of-the-art *DFS-Interleave* algorithm [1].

*Find-Next-*`YES`*-Concept Subroutine.* For ease of presentation, we first define a sub-routine, called *find-next-*`YES`*-concept* $(\varphi, u)$, for a given entity $\varphi$ and a `YES`-concept $u$ for $\varphi$, to return either:

- the *first* `YES`-concept out-neighbour $v$ of $u$ if $v$ exists, or
- `NULL` (in this case, $u$ is the target concept $t^*_\varphi$).

Specifically, it works as follows:

- if $u$'s out-degree $d_u = 0$, return `NULL`;
- otherwise, for each out-neighbour $v$ of $u$:
  - ask the oracle with question $q(\varphi, v)$; if $q(\varphi, v) = 1$, return $v$;
- return `NULL`.

In the worst case, the query cost of *find-next-*`YES`*-concept* $(\varphi, u)$ is $d_u \in O(d)$, where $d$ is the maximum out-degree of the nodes in $H$. In the rest of this paper, we refer to the *find-next-*`YES`*-concept* sub-routine as the *find-next* sub-routine for short.

**The Straightforward Baseline Algorithm.** Without loss of generality, we assume that the *root* of $H$ is a `YES`-concept for the given query entity $\varphi$, because otherwise, the target concept of $\varphi$ does not exist. The basic idea of this baseline algorithm is to "descend" a path from the root to the target concept $t^*_\varphi$ step by step.

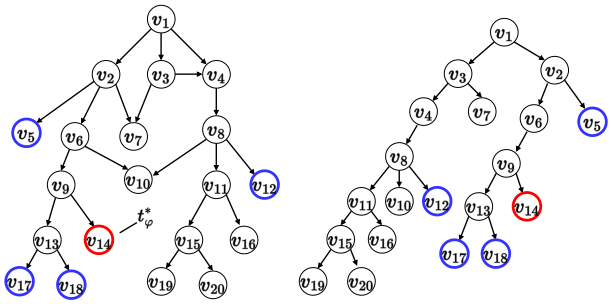
Specifically, starting with $u$ being the root of $H$, the baseline algorithm works as follows:

- $v \leftarrow$ *find-next-*`YES`*-concept* $(\varphi, u)$
- if $v$ is `NULL`, return $u$ as the target concept $t^*_\varphi$;
- otherwise, $u \leftarrow v$, and repeat the above steps;

*Analysis.* It is easy to verify that the overall query cost of this straightforward algorithm is bounded by $O(d \cdot h)$.

**The *DFS-Interleave* Algorithm [1].** The number of times invoking the *find-next* sub-routine is the efficiency bottleneck, and the core design of the *DFS-Interleave* algorithm is to reduce it from $O(h)$ to $O(\log n)$, where $n = |V|$ is the number of concepts in $H$.

To achieve this, the *DFS-Interleave* algorithm first performs a *depth-first search* (DFS) on $H$ starting from the root. However, this DFS favors selecting the most *out-reaching unvisited* out-neighbour during the search. Here, the *out-reachability* of a node $u$ is defined as, at the current moment during the DFS, the number of unvisited nodes that $u$ can reach in $H$. The resultant DFS-tree is called the *DFS heavy-path tree*, denoted by $T$. Moreover, the *first* child node (in the above DFS visiting order) of a node $u$ in $T$ is called the *left-most child* of $u$, and a path from $u$ to $v$ in $T$ that traverses the left-most child nodes only is called a *left-most path* from $u$ to $v$.




(a) A concept hierarchy $H$ (b) A DFS Heavy-path Tree $T$

Fig. 2. A hierarchy $H$ and its DFS heavy-path tree $T$, where $v_{14}$ is the target concept $t^*_\varphi$. As will be discussed in Section IV, the nodes in blue are concepts with examples similar to $\varphi$.

**Example 1.** *Figure 2 (a) shows an example concept hierarchy $H$. In the DFS from $v_1$, at this moment $v_1$ is the only* visited *node. The out-reachabilities of $v_2$, $v_3$ and $v_4$ are respectively* 10*,* 11*, and* 9 *(including the node itself). Thus, at this moment, $v_3$ is selected as the* first *child of $v_1$ to proceed with the DFS. Figure 2(b) shows the resultant DFS heavy-path tree $T$ of $H$, where the path from $v_1$ to $v_{19}$, the path from $v_2$ to $v_{17}$, and each leaf node (not on those paths already, e.g., $v_7$ and $v_5$) are all left-most paths.*

Consider a DFS heavy-path tree $T$; the nodes on any path from the root to some node $u$ in $T$ can be *covered* by a *sequence* of left-most paths in $T$. For example, in Figure 2 (b), all the nodes on the path from $v_1$ to $v_{14}$ in $T$ are covered by three left-most paths: $v_1 \rightsquigarrow v_{19}$, $v_2 \rightsquigarrow v_{17}$ and $v_{14}$. Therefore, the basic idea of the *DFS-Interleave* algorithm is to interleave *binary search* on left-most paths to find the deepest `YES`-concept $u$ and the *find-next* sub-routine on $u$ to "jump" to the next left-most path.

*The DFS-Interleave Algorithm.* Given a query entity $\varphi$ and the DFS heavy-path tree $T$ of $H$, starting with $u$ being the root of $T$, the *DFS-Interleave* algorithm, works as follows:

Step 1. $\pi \leftarrow$ the left-most path from $u$ to the leaf node;
Step 2. perform a *binary search* on $\pi$ to find the *deepest* `YES`-concept $v$ on $\pi$;
Step 3. $u \leftarrow$ *find-next-*`YES`*-concept* $(\varphi, v)$;
Step 4. if $u$ is `NULL`, then return $v$ as the target concept $t^*_\varphi$; otherwise, repeat the above steps.

*Correctness.* [1] proved that for any query entity $\varphi$, if the target concept $t^*_\varphi$ exits in $H$, then the *DFS-Interleave* algorithm on the DFS heavy-path tree $T$ of $H$ can correctly identify $t^*_\varphi$.

*Analysis.* Consider every execution of the four steps in *DFS-Interleave* as an *iteration*. By the known result of the *heavy path decomposition* [13], Tao et al. showed the following fact:

**Fact 1** ( [1]). *The total number of iterations of* DFS-Interleave *is at most* $1 + \lfloor \log_2 n \rfloor \in O(\log n)$.

In each iteration, the binary search on a left-most path $\pi$ takes $O(\log h)$ questions, and the *find-next* sub-routine takes $O(d)$ questions. Combing with Fact 1, the overall query cost of *DFS-Interleave* is bounded by $O(\log n \cdot \log h + d \cdot \log n)$. By more careful analysis, Tao et al. tighten the query cost bound to be $O(\log n \cdot \log h + d \cdot \log_d n)$. Moreover, a lower bound for the IGS problem is shown:

**Fact 2** (Lower Bound [1]). *Given a concept hierarchy $H$ with $n$ nodes and maximum out-degree $d$, any correct algorithm for solving the IGS problem must ask at least $(d-1) \cdot \lfloor \log_d n \rfloor$ questions.*

**Limitations and Challenges.** Despite the nearly optimal theoretical bound, the actual query cost of *DFS-Interleave* is still less satisfying. This is because, while it reduces the number of invocation of the *find-next* sub-routine from $O(h)$ to $O(\log n)$, this number is unfortunately still too large in practice. To see this, if the invocation happens to be on a node $u$ with a large out-degree, then the query cost can be as large as $d_u$. Even worse, in practice, concept hierarchies are often *short* and *flat* (i.e., they do not have too many levels but their nodes do have large degrees). A concept (e.g., `animal`) usually has many sub-concepts (e.g., various kinds of animals), but the level of the concepts would not be too deep. Indeed, we made consistent observations on the real datasets in our experiments. Taking the YAGO3 dataset as an example again; it has $h = 20$ yet $d = 44,983$ out of $n = 568,104$ nodes, on which *DFS-Interleave* takes $2,484$ questions on average per query entity.

Therefore, reducing the number of invocations of the *find-next* sub-routine on the nodes with large out-degrees is of great importance to practical query cost. As we will detail in Section IV, we address this challenge by additionally exploiting the *knowledge of the query entities*.

### C. Result-Sensitive Binary Search

Before we reveal the details of our proposed algorithms, let us describe the last piece of preliminary about *result-sensitive binary search* (a.k.a., *unbounded binary search*) [14] for solving the following search problem in the context of IGS:

**Definition 2** (Deepest `YES`-Concept Search). *Given a path $\pi$ of length $L$ in the concept hierarchy $H$, where $\pi[1]$, the first node on $\pi$, is a `YES`-concept, the goal is to find the deepest `YES`-concept on $\pi$ through interactions with the oracle.*

Clearly, this problem can be solved by a standard binary search with a query cost complexity bounded by $O(\log L)$. Next, we introduce the *result-sensitive binary search* which can achieve a better bound in certain cases.

In what follows, we use $\pi[i]$ to denote the $i^{\text{th}}$ node on $\pi$, where $1 \le i \le L$, and let $\ell$ be the index of the deepest `YES`-concept on $\pi$, namely, $\pi[\ell]$ is the node to locate for the above search problem.

**Result-Sensitive Binary Search.** The query cost of the result-sensitive binary search is bounded by $O(\log \ell)$ slightly improving the bound $O(\log L)$ of the standard binary search. Specifically, the algorithm works in two phases:

Phase 1. check the nodes on $\pi$ at index $2^i$, for $i = 1, 2, \ldots, \lfloor \log_2 L \rfloor$, one by one until either the first `NO`-concept is found at index $a = 2^i$ for some $i$ or all these $\lfloor \log_2 L \rfloor$ nodes are found to be `YES`-concept in which case set $a = L$.

Phase 2. perform a standard binary search on the sub-path $\pi[a/2, a]$, which contains all the nodes on $\pi$ from $\pi[a/2]$ to $\pi[a]$, to find the deepest `YES`-concept.

*Correctness.* It can be verified that $\pi[\ell]$ must be on the sub-path $\pi[a/2, a]$. And thus, the correctness follows immediately.

*Query Cost Analysis.* First, in Phase 1, in either case, at most $O(\log a)$ questions are asked. By the fact that $\ell \ge a/2$, the query cost is thus bounded by $O(\log \ell)$. Second, in Phase 2, there are at most $a - a/2 = a/2 \le \ell$ nodes on the sub-path $\pi[a/2, a]$. The query cost of the standard binary search is bounded by $O(\log \ell)$. Therefore, the overall query cost of the result-sensitive binary search is bounded by $O(\log \ell)$. We summarize this in the following lemma:

**Lemma 1.** *The query cost of the result-sensitive binary search is bounded by $O(\log \ell)$, where $\ell$ is the index of the deepest `YES`-concept on the path $\pi$.*

## III. An Improved Algorithm

In this section, we propose a modified version of *DFS-Interleave*, called *Target-Sensitive IGS* (*TS-IGS*). The basic idea of our *TS-IGS* algorithm is merely to replace the standard binary search in Step 2 of *DFS-Interleave* with a *result-sensitive binary search*. With a careful analysis, we prove that the query cost complexity of *TS-IGS* is bounded $O(\log n \cdot \log(\frac{L}{\log n}) + d \cdot \log_d n)$, where $L$ is the length of the path from the root of the DFS heavy-path tree $T$ to the target concept $t^*_\varphi$. Moreover, we show that when $L \in O(\log n)$ (which is often the case in practice), the query cost of *TS-IGS* is bounded by $O(d \cdot \log_d n)$, which is *optimal* up to a constant factor; and when $L \in O(\text{polylog } n)$, the query cost of *TS-IGS* is $O(\log n \log\log n + d \cdot \log_d n)$, which is still strictly better than the state-of-the-art bound of *DFS-Interleave*, given that $h$ can be as large as $n$ in the worst case.

### A. Our Target-Sensitive IGS Algorithm

**An Overview of *TS-IGS*.** Our *TS-IGS* algorithm also runs on the DFS heave-path tree $T$ of the concept hierarchy $H$ as introduced in Section II-B. The only difference between *TS-IGS* and *DFS-Interleave* lies in how they perform a search to find the deepest `YES`-concept on the left-most paths $\pi$. Specifically,

> *TS-IGS* replaces the standard binary search in Step 2 of *DFS-Interleave* with a result-sensitive binary search.

*Correctness.* The correctness of *TS-IGS* immediately follows the correctness of result-sensitive binary search and *DFS-Interleave*.

**Query Cost Analysis.** In what follows, we denote the path in $T$ from the root to $t^*_\varphi$ by $\pi_{t^*_\varphi}$ and the length of $\pi_{t^*_\varphi}$ by $L$. Let $x \geq 1$ be the total number of iterations performed in *TS-IGS*.

**Lemma 2.** *The total number of iterations $x$ is at most $L$.*

*Proof.* The lemma follows from the fact that in each iteration, *TS-IGS* must "descend" at least one step along the path $\pi_{t^*_\varphi}$. □

**Lemma 3.** *The query cost of* TS-IGS *is bounded by* $O(x \cdot \log \frac{L}{x} + d \cdot \log_d n)$.

*Proof.* It suffices to bound the overall query cost of the result-sensitive binary search in Step 2 of the algorithm description of *DFS-Interleave* in Section II-B because the overall cost of the *find-next* sub-routine is irrelevant.

Let $\ell_i \geq 1$ be the index of the deepest YES-concept on the left-most path $\pi_i$ at the $i^{\text{th}}$ iteration for $i = 1, 2, \ldots, x$. Observe that all the YES-concepts on the sub-path $\pi_i[1, \ell_i]$ of the left-most path $\pi_i$ must be on the path $\pi_{t^*_\varphi}$. As a result, we have $\sum_{i=1}^{x} \ell_i \leq L$.

According to Lemma 1, the overall query cost of all the result-sensitive binary searches of these $x$ iterations can be calculated as (the constant factors are omitted here for succinctness):

$$\sum_{i=1}^{x} \log_2 \ell_i \leq x \cdot \log_2 \left( \frac{\sum_{i=1}^{x} \ell_i}{x} \right) \leq x \cdot \log_2 \frac{L}{x},$$

where the first inequality comes from the *concavity* of the logarithmic function. □

Let $K = 1 + \lfloor \log_2 n \rfloor$; by Fact 1, we know that the number of iterations $x$ also satisfies $1 \leq x \leq K$.

**Lemma 4.** *Consider function $f(x) = x \cdot \log_2 \frac{L}{x}$, where $1 \leq x \leq \min\{K, L\}$; we have the following inequalities:*

- *when $L > e \cdot K$, $f(x) \leq K \cdot \log_2 \frac{L}{K}$;*
- *when $L \leq e \cdot K$, $f(x) \leq K \cdot \log_2 e$.*

*Proof.* It can be verified that the derivative of $f(x)$ is $f'(x) = \frac{1}{\ln 2} \cdot (\ln \frac{L}{x} - 1)$. We consider the following three cases:

- Case 1: when $e \cdot x \leq e \cdot K < L$, we have $f'(x) \geq 0$, and thus,
$$f(x) \leq f(K) = K \cdot \log_2 \frac{L}{K}.$$
- Case 2: when $e \cdot x \leq L \leq e \cdot K$, we have $f'(x) \geq 0$, and thus,
$$f(x) \leq f(\frac{L}{e}) = \frac{L}{e} \cdot \log_2 e \leq K \cdot \log_2 e.$$
- Case 3: when $L < e \cdot x \leq e \cdot K$, we have $f'(x) < 0$, and thus,
$$f(x) \leq f(\frac{L}{e}) = \frac{L}{e} \cdot \log_2 e \leq K \cdot \log_2 e.$$

The lemma follows. □

Next, we prove the following theorem:

**Theorem 1.** *For a DFS heavy-path tree $T$ with maximum out-degree $d \geq 2$, namely, $T$ is not a single path, let $L$ be the length of $\pi_{t^*_\varphi}$ in $T$. The query cost* TS-IGS *is bounded by* $O(\log n \cdot \log(\frac{L}{\log n}) + d \cdot \log_d n)$.

*Proof.* When $L \in \omega(\log n)$, we have $L > e \cdot K$; by Lemma 4 and Lemma 3, the query cost bound $O(\log n \cdot \log \frac{L}{\log n} + d \cdot \log_d n)$ follows. When $L \leq e \cdot K$, we have $f(x) \leq K \cdot \log_2 e \in O(\log n)$, thus, the query cost is bounded by $O(\log n + d \cdot \log_d n) = O(d \cdot \log_d n)$. This is because, for $d \geq 2$, we have $(d-1) \cdot \log_d n = \frac{d-1}{\log_2 d} \cdot \log_2 n \geq \log_2 n$.

When $e \cdot K < L \in O(\log n)$, we have $f(x) \leq K \cdot \log_2 \frac{L}{K} \in O(\log n)$. Thus, the query cost is also bounded by $O(d \cdot \log_d n)$.

Therefore, Theorem 1 follows. □

**Remark 1.** The key property of our *TS-IGS* algorithm is that its query cost is *target sensitive* rather than depending on the length $h$ of the longest path in $T$. In other words, if $t^*_\varphi$ is not that deep in $T$, it can be found by *TS-IGS* efficiently, and the cost can be optimal (up to a constant factor) when $L \in O(\log n)$ which is often the case in practice because the concept hierarchy $H$ is often short and flat.

**Remark 2.** As we will show next, our *Example-Guided IGS* (*EG-IGS*) algorithm can ask just a few promising questions to locate a YES-concept $u$ which is not far from $t^*_\varphi$. Then, running *TS-IGS* starting from $u$ on the sub-tree $T_u$ rooted at $u$, the length $L$ with respect to $T_u$ will be relatively small. According to our experimental results, on average, $L \leq 14$ across all six real datasets of different types of data including images, texts, and gene sequences. In this case, the query cost is bounded by $O(d \cdot \log_d n)$ which is optimal.

## IV. An Example-Guided IGS Algorithm

While both *TS-IGS* and *DFS-Interleave* achieve nearly optimal theoretical query cost bounds by exploring the *domain knowledge* of the concept hierarchy $H$ (to construct the DFS heavy-path tree $T$ and then search on $T$), interestingly, they have not exploited any domain knowledge of another important factor in the IGS problem – the query entities – in their algorithm designs! In this section, we propose an extremely efficient algorithm, called *Example-Guided IGS* (*EG-IGS*), for solving the IGS problem. As its name suggests, our *EG-IGS* leverages the knowledge of the query entities to ask "*promising*" questions guided by *examples*.

### A. Algorithm Overview

Our *EG-IGS* also works with the DFS heavy-path tree $T$ of $H$.

**Examples of a Concept.** Consider a concept $u \in T$; an *example* of $u$ is an entity of which $u$ is the target concept in $T$. Examples of a certain concept can be either from an *external source* or from the query entities in history.

Notably, the acquisition of the examples can be vastly different for different types of data and heavily depend on the application scenarios. To avoid unnecessary distraction, in this section, we assume that each concept $u \in T$ is associated

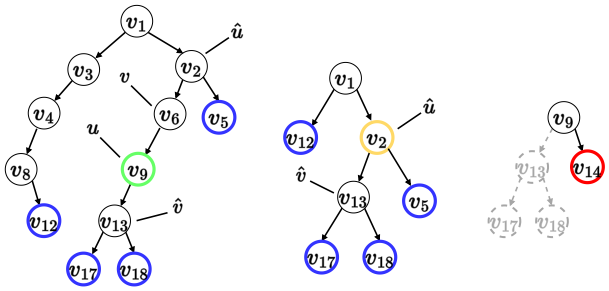



Fig. 3. In (a) and (b), $T_Q$ and $\hat{T}_Q$ are the deepest YES-concept in $T_Q$ and the corresponding compressed tree $\hat{T}_Q$, respectively; (c) shows the (pruned) sub-tree $T_u$ of $T$ shown in Figure 2(b).

with an example, and the *similarity* between entities can be measured properly, whose values are normalised in the range $[0, 1]$. The larger the similarity value is, the more similar the two entities are. In Section IV-C, we will describe the choices of similarity measurement between entities and explain how to obtain the examples *query cost free*.

**An Overview of Our *EG-IGS* Algorithm.** The rationale behind our *EG-IGS* algorithm is based on two crucial observations:

- *Crucial Observation 1:* Entities that belong to the same target concept should be similar to each other.
- *Crucial Observation 2:* The target concepts of similar entities should not be too far apart from each other in $T$, in the sense that the paths from the root to their target concepts would overlap with a considerably large portion.

Motivated by these two observations, at a high level, given a query entity $\varphi$, the *EG-IGS* algorithm works in four steps:

Step 1. find a set $\mathcal{E}$ of examples which are similar to $\varphi$;
Step 2. derive a set $Q$ of "promising questions" (or equivalently, the corresponding concepts for the questions) from $\mathcal{E}$;
Step 3. find the deepest YES-concept $u \in Q$;
Step 4. invoke *TS-IGS* algorithm on the sub-tree $T_u$ rooted at $u$ in $T$ to find the target concept $t^*_\varphi$.

### B. Detailed Implementation of EG-IGS

Next, we introduce the details of these four steps one by one.

**Step 1: Finding Similar Examples.** Given a query entity $\varphi$, the similarity between $\varphi$ and an example $p$ is denoted by $sim(\varphi, p)$. *EG-IGS* performs a *similarity search* to find a set $\mathcal{E}$ of examples of the concepts in $T$ with two sub-steps:

- $\mathcal{E} \leftarrow \{\text{example } p \mid sim(\varphi, p) \geq \tau\}$, where $0 \leq \tau \leq 1$ is the similarity threshold;
- if $|\mathcal{E}| > k$, then just keep the top-$k$ examples in $\mathcal{E}$, which are the most similar to $\varphi$, where $k \geq 1$ is a *constant* integer.

In our experiments, setting $\tau = 0.6$ and $k = 20$ works well across all six real datasets. The size of $\mathcal{E}$ is thus at most $k \in O(1)$, a constant.

**Step 2: Deriving Promising Questions.** By definition, each example (i.e., entity) $p \in \mathcal{E}$ corresponds to a concept $t^*_p \in T$. Denote the root of $T$ by $r_T$. *EG-IGS* collects all the nodes on the path from $r_T$ to $t^*_p$ as *promising questions* into the question set $Q$, for all $p \in \mathcal{E}$.

Since the question set $Q$ essentially is a set of all the nodes on at most $k$ paths from $r_T$, it can be verified that $Q$ itself forms a sub-tree of $T$ rooted at $r_T$. We denote this sub-tree by $T_Q$ and refer to it as the *promising question tree* for $\varphi$.

*A Running Example.* In Figures 2 and 3, the concepts with examples similar to $\varphi$ are highlighted in blue. Figure 3(a) shows the corresponding promising question tree $T_Q$. □

**Lemma 5.** *The promising question tree $T_Q$ contains at most $k \cdot h \in O(h)$ nodes, has a height of at most $h$, and has a maximum out-degree of at most $k \in O(1)$.*

*Proof.* This lemma follows from the fact that $T_Q$ consists of $k$ paths starting from $r_T$, and each path's length is at most $h$. □

**Step 3: Finding the Deepest YES-Concept $u$ in $T_Q$.** Naively, the deepest YES-concept in $T_Q$ can be found by simply invoking our *TS-IGS* algorithm. By Theorem 1 and Lemma 5, the query cost of this naive approach is bounded by $O(\log h \cdot \log \frac{h}{\log h} + k \cdot \log_k h) \subseteq O(\log^2 h)$. Next, we show a more sophisticated algorithm, called *Compress-and-Find*, to achieve a query cost bound $O(\log h)$.

It is worth mentioning that improving the $O(\log^2 h)$ bound to $O(\log h)$ is of great importance because the latter can be subsumed by the term $O(d \cdot \log_d n)$ while the former cannot. This fact is crucial to the establishment of optimality of our *EG-IGS* in certain cases.

*Our Compress-and-Find Algorithm*, namely *CaF*, works as follows:

1) Compress the promising question tree $T_Q$ by: (i) keeping the root, all the leaf nodes, and all the nodes with out-degrees $\geq 2$ in $T_Q$, and (ii) contracting all the single paths (containing only nodes with out-degree=1) between those nodes kept above. Denote the resultant *compressed tree* of $T_Q$ by $\hat{T}_Q$.
   Figure 3(b) shows the compressed tree $\hat{T}_Q$ of the promising question tree $T_Q$ shown in Figure 3(a), where the single paths $v_3 \rightsquigarrow v_8$ and $v_6 \rightsquigarrow v_9$ are compressed.
2) Invoke the straightforward baseline algorithm (introduced in Section II-B) on $\hat{T}_Q$ to find the deepest YES-concept $\hat{u}$. Continue the previous example; as $t^*_\varphi = v_{14}$, the deepest YES-concept $\hat{u}$ in $\hat{T}_Q$ in Figure 3(b) is $v_2$.
3) If $\hat{u}$ is a leaf node in $\hat{T}_Q$, then return $\hat{u}$ as the deepest YES-concept $u$ in $T_Q$ since $\hat{u}$ must also be a leaf node of $T_Q$.
4) Otherwise, let $v \leftarrow$ *find-next-YES-concept* $(\varphi, \hat{u})$ in (uncompressed) $T_Q$ with $\hat{u}$'s out-neighbours ordered by similarity to $\varphi$ from large to small.
   a) If $v$ is NULL, return $\hat{u}$ as the deepest YES-concept $u$ in $T_Q$.
   b) Otherwise, let $\hat{v}$ be the child node of $\hat{u}$ in the compressed tree $\hat{T}_Q$, which is reachable from $v$ in $T_Q$. In other words, $v$ is on the single path from $\hat{u}$ to $\hat{v}$

in $T_Q$. Observe that $\hat{v}$ must be a NO-concept because otherwise, $\hat{u}$ would not have been returned. As $v$ is a YES-concept and the path from $v$ to $\hat{v}$ in $T_Q$ is a single path, the deepest YES-concept $u$ on this path can be found by a standard binary search.

Continue the example in Figure 3; the *find-next* sub-routine on $\hat{u} = v_2$ returns a YES-concept $v = v_6$. We know that the child node $\hat{v} = v_{13}$ of $v_2$ in $\hat{T}_Q$ is a NO-concept. Therefore, the deepest YES-concept $u = v_9$ in $T_Q$ must be on the path from $v_6$ to $v_{13}$.

*Query Cost Analysis of Step 3.* Next, we prove that the query cost of the above *CaF* algorithm is bounded by $O(\log h)$.

**Lemma 6.** *The compressed tree $\hat{T}_Q$ has $O(1)$ nodes.*

*Proof.* According to the contraction rule of $\hat{T}_Q$, there are at most $k$ leaf nodes, and every internal node in $\hat{T}_Q$ must have a degree of at least 2. Observe that we can transform $\hat{T}_Q$ into a *full* binary tree $\mathcal{T}$ (where every node in $\mathcal{T}$ has either zero or two child nodes) by adding internal nodes only (to decrease the out-degrees of those nodes $u$ with $d_u > 2$), while keeping the number of leaf nodes unchanged. We use $|X|$ to denote the number of nodes in a tree $X$. Thus, we have $|\hat{T}_Q| \leq |\mathcal{T}|$. Moreover, it is known that the *perfect* binary tree has the largest number of nodes among all possible full binary trees with the same number of leaf nodes. Consider a perfect binary tree $\mathcal{T}^*$ on $k \leq 2^{\lceil \log_2 k \rceil} \leq 2 \cdot k$ leaf nodes; there are at most $4 \cdot k$ nodes in $\mathcal{T}^*$. Therefore, $|\hat{T}_Q| \leq |\mathcal{T}| \leq |\mathcal{T}^*| \leq 4 \cdot k \in O(1)$. □

**Lemma 7.** *The query cost bound of the* CaF *algorithm is* $O(\log h)$.

*Proof.* Clearly, Steps (1) and (3) of the CaF algorithm are query cost-free. By Lemma 6, the query cost of running the straightforward baseline algorithm on $\hat{T}_Q$ in Step (2) is $O(1)$. By Lemma 5, the cost of invoking the *find-next* sub-routine in Step (4)(a) is also $O(1)$, and the cost of the standard binary search on the single path in Step (4)(b) is bounded by $O(\log h)$. □

**Step 4: Finding the Target Concept $t^*_\varphi$ in $T_u$.** Given the deepest YES-concept $u$ in $T_Q$, *EG-IGS* invokes the *TS-IGS* algorithm (in Section III-A) on the sub-tree $T_u$ rooted at $u$ of the DFS heavy-path tree $T$, and returns the target concept $t^*_\varphi$ thus found.

Continue with the example in Figure 3; the sub-tree $T_u$ is shown in Figure 3(c), which can be pruned by the fact that $v_{13}$ is a NO-concept.

*Query Cost Analysis of Step 4.* Let $L'$ be the length of the path from $u$ to $t^*_\varphi$ in $T_u$. Clearly, $L' \leq L$. For example, in Figure 3(c), $L' = 1$. According to Theorem 1, the query cost of *TS-IGS* on $T_u$ is bounded by as follows:

- when $L' \in O(\log n)$, $O(d \cdot \log_d n)$.
- when $L' \in \omega(\log n)$, $O(\log n \cdot \log \frac{L'}{\log n} + d \cdot \log_d n)$.

**The Overall Query Cost of *EG-IGS*.** Observe that Steps 1 and 2 are query cost-free, as no questions are needed to find similar examples or to compute the promising question tree $T_Q$. From the query cost analysis of Steps 3 and 4, we have the following theorem:

**Theorem 2.** *The overall query cost of* EG-IGS *is bounded by:*

- *when $L' \in O(\log n)$, $O(d \cdot \log_d n)$;*
- *when $L' \in \omega(\log n)$, $O(\log n \cdot \log \frac{L'}{\log n} + d \cdot \log_d n)$.*

**A Discussion on the Superiority of *EG-IGS*.** In theory, the query cost bound of *EG-IGS* is slightly finer-grained than that of *TS-IGS*. In certain cases, it is optimal and strictly outperforms the bound of *DFS-Interleave*. However, the true superiority of our *EG-IGS* lies in its practical performance. According to the two crucial observations, under the guidance of the similar examples, the target concept $t^*_\varphi$ can often be captured in the promising question tree $T_Q$. Thus, after spending $O(\log h)$ questions in Step 3, $t^*_\varphi$ will be found, and the next step is merely to confirm $t^*_\varphi$ (by invoking the *find-next* sub-routine). Moreover, it is also often the case that $t^*_\varphi$ is a leaf node in $T$ and hence, in this case, $t^*_\varphi$ can be confirmed with zero query cost. On the other hand, even when $t^*_\varphi$ is not captured in $T_Q$, the deepest YES-concept $u$ is often not far from $t^*_\varphi$, and thus, the length $L'$ is small. According to our experimental results, $L' \leq 14$ holds for all cases in the experiments across six real datasets. In other words, in this case, *EG-IGS* is actually optimal in practice.

### C. Two Missing Pieces

In this subsection, we discuss the last two *missing pieces* in our example-guided technique: (i) how to obtain examples, and (ii) how to compute similarity between entities.

**How to Obtain Examples.** For different application scenarios, the entities can be different types of data, and the way to obtain examples for the concepts can be vastly different. However, the core idea to obtain examples is the same: *find examples from external sources!* The key to the success of this idea actually stems from the following interesting observation:

> *While labelling an entity with its target concept in $H$ may be challenging, finding an entity with a specified concept label can be much simpler.*

Motivated by this observation, we *search* examples for the concepts in $H$ from external sources.

*Obtaining Examples in Image Classification Tasks.* To illustrate, let us take the image classification task as an example. For each concept $u \in H$, we obtain examples for $u$ by as simple as searching the keyword of concept $u$ with the Google Images search engine, and then taking the first image in the search result as an example for $u$.

There are two nice advantages of the above approach for obtaining examples from external sources:

1) This approach can be made automatic with computer programs.
2) This approach is completely *query cost free*.

Our experimental results show that examples from external sources work pretty well for our *EG-IGS* algorithm. More details about how we obtain examples can be found in Section VI and in Table I.

*What If External Examples Are Not Available?* In this case, we accumulate examples by recording each query entity when we perform IGS for them. In our experiments, after around 250 queries (from the cold start) the query cost reduces substantially.

**How to Measure Similarity.** Analogous to how we obtain examples, to measure the similarity between entities, we adopt existing pre-trained models, and normalise the similarity scores into real numbers in the range $[0, 1]$. Continue with the image classification task example, we adopt the pre-trained image embedding network of Google TensorFlow [15] to compute image embeddings and utilise cosine similarity between the embeddings to measure the similarity between the images. More details on the similarity measurement for other types of datasets can be found in Section VI.

**Remark.** It is worth mentioning that, the above approaches are just some of many possible ways to obtain examples and measure entity similarity for the purpose of showing the feasibility of our ideas. There might be better approaches for these tasks, yet the discussion on these approaches is out of the scope of this work.

## V. Discussions & Other Related Work

### A. Further Discussions

**Single Target v.s. Multiple Targets.** As astute readers may have noticed, in the IGS problem formulation, the target concept is unique, while, in some applications, query entities may admit multiple target concepts.

First, we emphasize that the single-target IGS problem is already useful for a vast number of applications, especially for those in scientific scenarios that require high-accuracy labelling, e.g., labelling disease genes or virus. These tasks often require high costs for each question, as they have high standards for the oracles which are often experts and scientists. Therefore, reducing the query cost is of great importance, and our proposed *EG-IGS* algorithm has significant value for these important application scenarios.

Second, our *EG-IGS* can also handle the multi-target problem, simply by the following steps:

- run *EG-IGS* to find an arbitrary target concept $t^*$ as if the query entity has a unique target;
- recursively solves the multi-target problem on the subtree rooted at each of the `YES`-concept on the root-to-$t^*$ path until no more `YES`-concept is found.

However, we note that the above algorithm is, unfortunately, heuristic for the multi-target problem. This is because, with the currently limited oracle capability, confirming the fact that all the targets have been found requires $\Theta(d \cdot Y)$ questions, where $Y$ is the number of `YES`-concepts induced by all the target concepts for the query. In this case, the brute-force algorithm is actually optimal. To admit more efficient algorithms, one may need a *more powerful oracle* for the (more challenging) multi-target problem, e.g., an oracle that can tell which child nodes of a given `YES`-concept are also YES-concepts. We leave the exploration on this direction as future work.

**Multiple Questions at a Time.** With an oracle that can answer $k$ questions at a time, our algorithms can improve the binary-search steps to $k$-ary searches and thus, the base of the logarithmic term in the query complexity can be improved from 2 to $k$.

### B. Other Related Work

We outline some existing work that are also relevant to the IGS problem below.

**MIGS.** [16] consider a multiple-choice question setting, where the oracle is given multiple nodes (rather than just one in IGS) in each question, and it returns the node that leads to the target concept if given. Their algorithm, MIGS, builds a decision tree based on the distribution of the target concepts to guide the selection of nodes to be asked in each iteration round. The nodes with a higher probability to be the target concept are asked first. They develop a model to estimate the target concept probability during the labelling task such that nodes that are more likely to be the target concept would have a higher probability to be asked.

**STBIS.** [17] consider a setting with multiple target concepts for one entity. They assume that the number of target concepts is unknown, and they impose a constraint on the number of interaction rounds. Under this constraint, their goal is to obtain a set $S$ of nodes that the average shortest-path distance in $H$ from each target concept to its nearest node in $S$ is as small as possible. The algorithm proposed by [17], STBIS, uses a greedy strategy to ask the node that has the most "*gain*" in each interaction round, where the gain is decided by the target concept distribution and the estimated distances between the node to the target concepts. We remove the constraint on the number of interaction rounds of STBIS in our experiments.

**BinG.** [18] consider a varying cost for each question. Their algorithm, BinG, asks a node that partitions the remaining graph into two most balanced halves in terms of the total "weight" of the nodes in each half. The weight of a node is formulated by its question cost and its probability of being the target concept. Furthermore, [19] address the IGS problem in a noisy setting, where the oracle may return incorrect answers with a known probability. They assume a known target concept distribution and use the Bayesian method [20] to infer the target concept from a set of nodes asked, together with the answers returned by the oracle. With the posterior probability, the node that maximizes the expectation of finding the target concept in each interaction round is selected in the question. As we consider deterministic oracles only in this paper, this noisy setting is irrelevant.

**Remark.** The above algorithms heuristically use target concept distribution to accelerate the search. In real-world applications, it is difficult to obtain a target concept distribution beforehand (e.g., every image dataset may have a different class distribution), while the distribution may change as more search is done. In comparison, it is easier to obtain external examples as done in our proposal, and our *EG-IGS* is robust to changes in the target concept distribution.

**Other Loosely Related Work.** The idea of the interactive graph search problem was first formulated by [21]. The focus and definition of [21] are quite different from the IGS problem studied in this paper. Specifically, [21] aims to select a number of questions in advance and asks these questions to the oracle in batch to reduce the search space for the target node. Moreover, while faceted search [22]–[24] and finding maximally specific sentences [25] share some similarity with the IGS problem, they are still different. In particular, faceted search is to filter search results based on multiple facets, and finding maximally specific sentences focuses on knowledge discovery model regarding theory extraction.

## VI. Experiments

**Datasets.** We deploy six real-world datasets covering three typical domains (images, texts, and gene sequences). Table I records their statistics.

**`ImageNet`** [26] is a collection of images, each affiliated with a class label. Each class label is a word from the WordNet hierarchy [27], serving as the input concept hierarchy $H$, such that each image belongs to a node in $H$. We sample uniformly at random $1,000$ images from its 2017 challenge dataset[2] as the query entities.

**`Coco`** [28] is another image dataset with class labels. We also use the WordNet hierarchy as the concept hierarchy $H$, by mapping the label names of the images to the nodes in the WordNet hierarchy with the same names. If an image in `Coco` has multiple class labels, we randomly select one as its target concept. We sample uniformly at random $1,000$ images from its 2014 dataset[3] as the query entities.

**`Amazon`** [29] is a product dataset from Amazon. Each product item comes with a textual description (i.e., sentences) and belongs to a category in a product category hierarchy, which serves as the concept hierarchy $H$. We randomly sample $1,000$ items from the dataset as the query entities.

**`YAGO3`** [30] consists of a category hierarchy and Wikipedia entities. Its category hierarchy, which is used as the concept hierarchy $H$, is constructed by combining the WordNet hierarchy and Wikipedia categories. When an entity belongs to multiple nodes in $H$, we randomly select one as its target concept. We randomly sample $1,000$ entities as the query entities and download a summary from the Wikipedia website for each entity as its description.

**`NCBI`** [31] has a taxonomy that contains names and phylogenetic lineages of different organisms. We use the taxonomy as the concept hierarchy $H$ and randomly select $1,000$ phylogenetic lineages as the query entities. For each entity sampled, we download a summary from the Wikipedia website as its description.

**`NCBIV`** [31] is a sub-tree of the NCBI gene sequence hierarchy rooted at node "virus", which is used as the concept hierarchy $H$. We use $1,000$ gene sequences from popular viruses as the query entities.

[2]https://image-net.org/challenges/LSVRC/2017/

[3]https://cocodataset.org/#download

TABLE I: Summary of Datasets

| Type | Dataset | $n$ | $d$ | $h$ | Example source |
|---|---|---|---|---|---|
| Image | `ImageNet` | 58,901 | 371 | 81 | Google Images |
| | `Coco` | 58,901 | 371 | 81 | Google Images |
| Text | `Amazon` | 26,170 | 77 | 9 | Amazon e-commercial |
| | `YAGO3` | 569,104 | 44,983 | 20 | Wikipedia |
| | `NCBI` | 2,426,961 | 43,706 | 37 | Wikipedia |
| Gene Seq. | `NCBIV` | 228,782 | 36,124 | 14 | NCBI database |

**Entity Similarity Measurement.** For the image datasets (`ImageNet` and `Coco`), we use the pre-trained image embedding network[4] of Google TensorFlow [15] to compute image embeddings and the cosine similarity as similarity measurement. For the text datasets (`Amazon`, `YAGO3`, `NCBI`), we use the *spaCy* Python package [32] for text similarity computation and re-scale the similarity score from $[-1, 1]$ to $[0, 1]$. For `NCBIV`, we deploy the edit distance between two gene sequences and normalise the distance into $[0, 1]$ by dividing the length of the longer sequence. The similarity score is computed as one minus the normalised edit distance. In all experiments, the default similarity threshold, $\tau$, for our *EG-IGS* is set to $0.6$.

### A. Methods for Comparison

**Our Algorithms.** We include three variants of our algorithms:

- *TS-IGS*: our target-sensitive IGS algorithm (Section III-A).
- *EG-IGS-H*: our *EG-IGS* algorithm with no external example source which accumulates examples as the queries are processed.
- *EG-IGS-E*: our full-gear *EG-IGS* algorithm with external examples. Specifically, the examples in each dataset are obtained in the same manner. For each concept $t$ in the concept hierarchy $H$, we use $t$ as a keyword to search in the corresponding example sources (see in Table I) of the dataset, and then take the first returned result as an example for concept $t$.

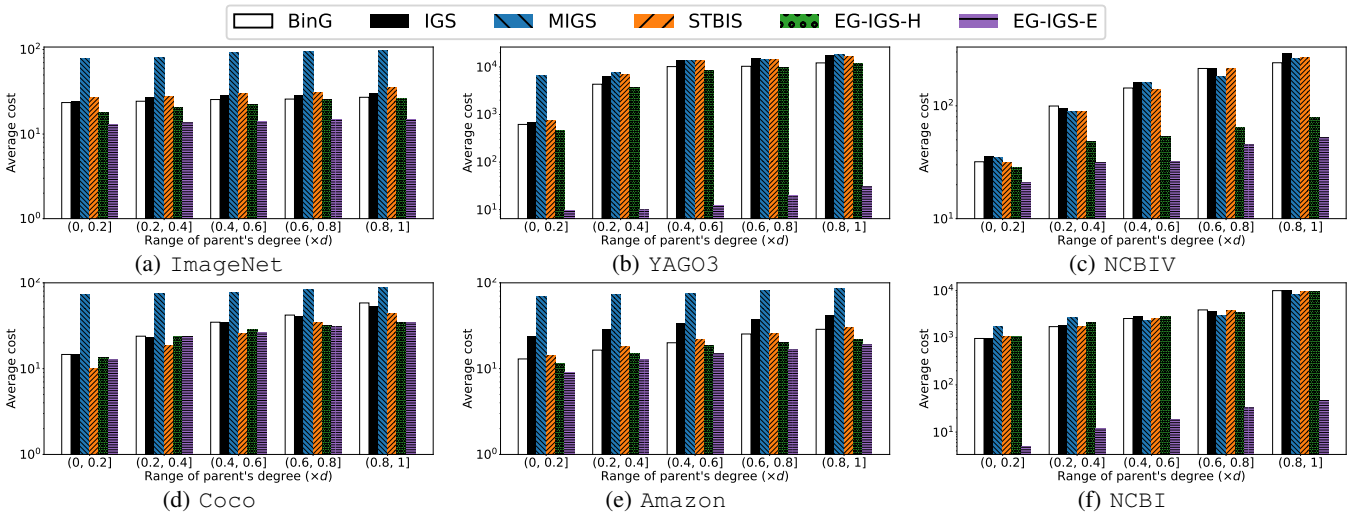
**Competitors.** We compare our algorithms with *four* state-of-the-art methods, which are elaborated in Section II and Section V-B. These methods were designed for either the IGS problem or its variants but can be adapted for our IGS problem straightforwardly. Specifically, they are: (i) IGS [1] which is the *DFS-Intervleave* algorithm introduced in Section II, (ii) BinG [18], (iii) STBIS [17], and (iv) STBIS [17]. Moreover, we note that BinG, STBIS, and MIGS are all based on target concept distribution. For STBIS and MIGS, we estimate the distribution as done in their original proposals. For BinG, we assume the distribution is unknown and apply their learning distribution on the fly strategy, that is to count the target node during the task.

### B. Comparison with the State of The Arts

We report the **average query cost** over $1,000$ query entities, i.e., the average number of questions asked per query entity.

**Overall Results.** As shown in Table II, *EG-IGS-E* incurs the lowest *cost* across all datasets, especially on those with large degrees (e.g. `YAGO3`, `NCBI` and `NCBIV` where $d > 30,000$).

[4]model URL: https://tfhub.dev/tensorflow/efficientnet/lite0/feature-vector/2




(a) ImageNet (b) YAGO3 (c) NCBIV

(d) Coco (e) Amazon (f) NCBI

Fig. 4: Average cost vs. the out-degree of the parent nodes of the target nodes

The gain in the cost for *EG-IGS-E* compared with its competitors, is at least 3.50% (25.57 vs 26.47 on Coco) and up to 560 times (13.66 vs. 7669.43 on YAGO3). This confirms that the guidance from examples is extremely effective and has provided strong knowledge of the target concepts.

Even without any external example, our *EG-IGS-H* algorithm still outperforms all competitors, with a gain ranging from 0.30% (26.39 vs. 26.47 on Coco) to 193.63% (54.30 vs 159.44 on NCBIV). An exception is observed on NCBI, where all the $1,000$ query entities happen to have *distinct* target concepts. In this case, the promising questions from examples accumulated from history queries can only help locate the parent of the target concept (at best). However, it happens that the nodes in NCBI have significantly large out-degrees, especially for those parents of the leaf nodes. As a result, a large number of questions have to be asked in the *find-next* sub-routine for the parent of the target concept at leaf level. This is true for all algorithms without external examples, including our *EG-IGS-H*.

Our *TS-IGS* has similar performance to IGS because its improvement mainly stems from the search on the left-most paths, while on the tested datasets, the query cost of the *find-next* sub-routine dominates the overall cost (detailed in Section VI-D). On Amazon which has the smallest $d$, our *TS-IGS* has an improvement of 12.15%.

Among all the competitors, IGS suffers when the hierarchy $H$ is *short* and *flat* (i.e., with small $h$ and large $d$). In this case, the left-most paths are short, and thus, no much gain from binary search on them (e.g., YAGO3, NCBI, and NCBIV). STBIS and BinG rely on a target concept distribution estimation to make the search "balanced" with the notions of the maximum closeness centrality (STBIS) and the potential candidate size (BinG). When the distribution estimation is less accurate (e.g. when the algorithms just started processing a few queries), they may trigger a high cost. Further, when the target concept distribution is *uniform* over all nodes in $H$, they barely obtain gains from the distribution. For MIGS, in its multiple-choice problem definition, the reachable child of a node can be found for one interaction (cost). Its optimization goal is to ask the node with the highest possibility first, regardless of the size of its children. Hence, MIGS might put the nodes with large out-degrees in priority, resulting in a relatively large cost in the binary question setting.

TABLE II: Average Cost for Different Algorithms

| | ImageNet | Coco | Amazon | YAGO3 | NCBI | NCBIV |
|---|---|---|---|---|---|---|
| BinG | **24.43** | 34.63 | **20.05** | **1795.89** | **2745.60** | **146.11** |
| IGS | 28.56 | 33.33 | 33.12 | 2484.47 | 2765.86 | 159.44 |
| STBIS | 29.32 | **26.47** | 21.87 | 2475.18 | 2751.56 | 149.15 |
| MIGS | 90.01 | 79.42 | 76.10 | 7669.43 | 2807.25 | 146.18 |
| Our methods | | | | | | |
| *TS-IGS* | 28.37 | 32.73 | 29.72 | 2484.21 | 2766.38 | 159.25 |
| *EG-IGS-H* | 15.40 | 26.39 | 17.27 | 1643.67 | 2765.80 | 54.30 |
| *EG-IGS-E* | **14.71** | **25.57** | **12.45** | **13.66** | **17.84** | **36.42** |

**Impact of the Parent's Out-degree of the Target Concept.** In the IGS problem, it is often the case that the target concept $t^*_\varphi$ is a leaf node in the hierarchy $H$. From our observation, the major query cost indeed comes from the *find-next* sub-routine running for the parent of $t^*_\varphi$, and thus, depends on the *parent's out-degrees*. Recall that the maximum out-degree of the nodes in $H$ is denoted by $d$. We divide the parents of $t^*_\varphi$'s into five categories in terms of their out-degrees: $(0, 0.2 \cdot d]$, $(0.2 \cdot d, 0.4 \cdot d]$, $(0.4 \cdot d, 0.6 \cdot d]$, $(0.6 \cdot d, 0.8 \cdot d]$ and $(0.8 \cdot d, d]$. For each category, we sample uniformly at random 200 concepts (if there are less than 200 concepts in the category, then take all of them) and use their corresponding examples as the query entities in each dataset.

Figure 4 reports the average query costs of all algorithms when varying the parent's out-degrees of $t^*_\varphi$'s on ImageNet, YAGO3, NCBIV. The average query cost increases with the parent's out-degrees, as the query cost of the *find-next* sub-routine increases. Although the impact of our algorithms *EG-IGS-H*, and *EG-IGS-E* is quite mild, they again outperform all competitors consistently in all categories. *EG-IGS-E*, in particular, retains a large performance gap–its average query costs only increase from 12.83 to 14.01 on ImageNet,

TABLE III: Target Concept Statistics

| | ImageNet | Coco | Amazon | YAGO3 | NCBI | NCBIV |
|---|---|---|---|---|---|---|
| $t^*_\varphi$ is a leaf | 51.40% | 14.60% | 82.00% | 99.30% | 99.90% | 30.10% |
| $t^*_\varphi \in T_Q$ | 68.70% | 81.70% | 98.60% | 99.90% | 100.0% | 97.90% |
| avg. $dist(u, t^*_\varphi)$ | 13.82 | 8.54 | 1.86 | 2.11 | 0 | 1.31 |

from 9.67 to 30.70 on YAGO3, from 20.74 to 51.99 on NCBIV, from 12.64 to 34.60 on Coco, from 9.11 to 19.24 on Amazon, and from 4.92 to 47.03 on NCBI from the categories with the smallest to the largest parent's out-degrees. Besides our two algorithms, BinG performs the best on ImageNet, YAGO3, NCBIV, Amazon, and NCBI, whose average query cost increases from 23.58 to 27.22, from 619.32 to 1,2174.55, from 31.93 to 240.83, from 12.97 to 28.72, and from 966.31 to 9978.26, respectively; STBIS performs best on Coco whose average cost increase from 10.11 to 44.33.

### C. Effectiveness of the Example Guidance

Next, we study the effectiveness of the guidance from examples. Table III shows the statistics of the target concept $t^*_\varphi$. The first row shows the percentage of $t^*_\varphi$ being a leaf node in $H$, reflecting how likely we can confirm $t^*_\varphi$ as the target concept with zero query cost. In Amazon, YAGO3 and NCBI, for more than 82% queries, $t^*_\varphi$ is a leaf, while in the other three datasets, this percentage is relatively low. The second row shows the percentage of queries with $t^*_\varphi \in T_Q$, directly reflecting the effectiveness of the guidance of examples. Specifically, except for the two imagery datasets ImageNet (68.7%) and Coco (81.7%), all other datasets have this percentage over 97.9%, indicating that in most cases the promising question tree $T_Q$ is able to capture $t^*_\varphi$. The third row reports the average distance (on $T$) from the deepest YES-concept $u \in T_Q$ to $t^*_\varphi$. It shows that even though $t^*_\varphi$ may not be captured in $T_Q$, things are not too bad. The average distance from $u$ to $t^*_\varphi$ is just at most 14 across all the datasets, implying that our *EG-IGS* is actually optimal on these datasets.

To further look into the effectiveness of the example guidance in reducing the invocations of the *find-next* sub-routine, which, as discussed earlier, the bottleneck in the query cost is. Table IV shows a breakdown of the average query costs of both IGS and *EG-IGS-E*. We can see that the average query costs spent on the *find-next* sub-routine of our *EG-IGS-E* are substantially smaller than those of IGS, and the only overhead is just the cost of searching on the promising question tree $T_Q$, which takes a small portion of the overall cost. Notably, on YAGO3 and NCBI, our *EG-IGS-E* dramatically reduces the *find-next* cost from over 2000 to less than 11, by just paying an overhead of less than 8 questions. To explain this, combining the statistic in Table III, we find that on these two datasets, $T_Q$ almost captures all $t^*_\varphi$ and $t^*_\varphi$ is a leaf node in over 99% of the cases. Hence, on these two datasets, *EG-IGS-E* has almost no need to invoke the expensive *find-next* sub-routine to confirm $t^*_\varphi$. Again, the powerfulness of the example guidance is thus evidenced.

TABLE IV: Average Cost Breakdowns

| | | ImageNet | Coco | Amazon | YAGO3 | NCBI | NCBIV |
|---|---|---|---|---|---|---|---|
| IGS | *Search on Paths* | 3.92 | 10.75 | 9.38 | 9.08 | 15.34 | 11.51 |
| | *find-next* | 24.66 | 22.58 | 23.74 | 2475.39 | 2750.52 | 147.93 |
| *EG-IGS-E* | *Search on $T_Q$* | 4.96 | 3.71 | 6.93 | 7.25 | 7.09 | 1.84 |
| | *Search on Paths* | 3.69 | 6.68 | 0.02 | 0 | 0 | 0.03 |
| | *find-next* | 6.06 | 15.18 | 5.50 | 6.41 | 10.75 | 34.55 |

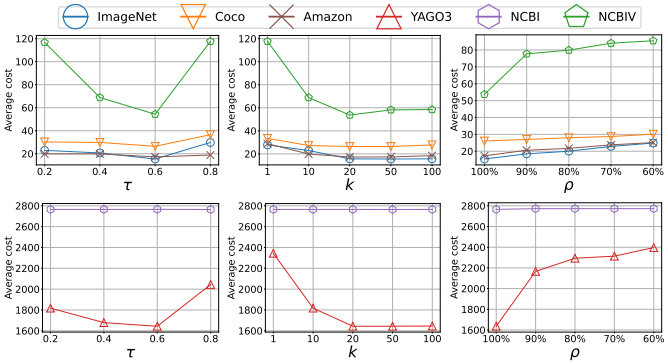



Fig. 5: Average cost vs. $\tau$, $k$, and $\rho$ (*EG-IGS-H*)

### D. Robustness of Example-Guided IGS

To study the robustness of our *EG-IGS* algorithms, we look into the impacts of the similarity threshold ($\tau$), the cap number ($k$) of the similar examples, the similarity measurement accuracy, the quantity and quality of the external examples, as well as cold start.

**Study 1: Impact of the Similarity Threshold $\tau$.** In our *EG-IGS* algorithms, an example is similar to the query entity $\varphi$ if its similarity to $\varphi$ is at least $\tau$. Next, we vary $\tau$ from 0.2 to 0.8. To amplify the impact of $\tau$, we set $k = 100$ (i.e., at most top-100 similar examples would be considered) in this experiment. Figures 5 and 6 show the average query cost of our two algorithms: *EG-IGS-H* and *EG-IGS-E*.

The average query costs of both algorithms decrease before $\tau$ reaches 0.6, followed by an increase afterward. This is because, when $\tau$ is too small, examples that are less similar to $\varphi$ will be considered, causing a larger promising question tree, and hence, more questions to ask. When $\tau$ grows further and becomes too large, it limits the number of examples that may be considered similar to $\varphi$, where the target concept $t^*_\varphi$ may be missed by the resultant promising question tree, and a less effective YES-concept is found in $T_Q$. This explains the increasing average query costs when $\tau > 0.6$ on YAGO3, Coco, NCBIV, and ImageNet for both algorithms. For the other datasets, as the entities are "denser" in their embedding space, good examples can still be captured even when $\tau > 0.6$. Thus, the average query costs of both algorithms have no obvious increase. Overall, the default $\tau = 0.6$ works well for both algorithms.

**Study 2: Impact of the Similar Example Number Cap $k$.** Next, we study the impact of $k$ by varying $k \in \{1, 10, 20, 50, 100\}$. Figures 5 and 6 show the results. When $k = 1$, it is likely that the target concept $t^*_\varphi$ is missed in $T_Q$ for both algorithms. The query cost immediately drops when $k$ increases to 10. After $k \geq 20$, both algorithms manage to

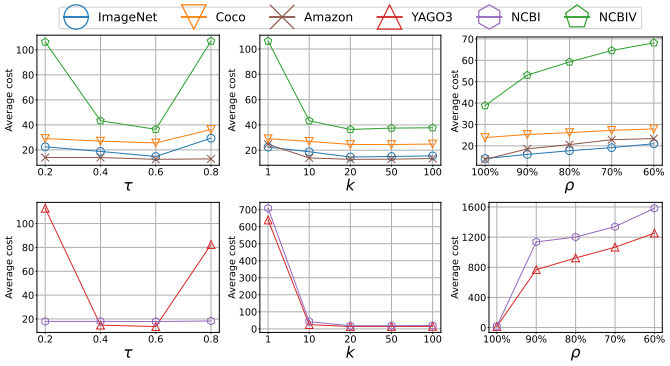



Fig. 6: Average cost vs. $\tau$, $k$, and $\rho$ (*EG-IGS-E*)

capture sufficient good examples, yet the query cost overhead barely increases, thanks to our efficient *Compress-and-Find* algorithm.

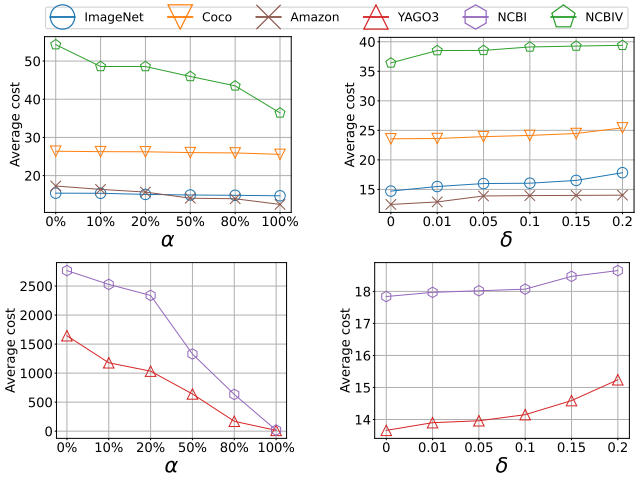
**Study 3: Impact of the Similarity Measurement Accuracy.** In this experiment, we study the case that the returned "similar" examples may not be reliable, i.e., they are not quite similar to $\varphi$. We introduce a real number parameter $\rho \in [0, 1]$, where with probability $\rho$, the true top-50 similar examples are returned, while with probability $1 - \rho$, 50 random examples (excluding the true top-50 ones) are returned as $\mathcal{E}$, the set of similar examples.

As shown in Figures 5 and 6, the average query costs of both *EG-IGS-H* and *EG-IGS-E* increase when $\rho$ decreases. This is expected as in this case, it is more likely that irrelevant examples to $\varphi$ are returned. Interestingly, comparing these results with those in Table II, we can see that even if with 40% of cases, similar examples are polluted ($\rho = 60\%$), *EG-IGS-E* outperforms all competitors on all datasets, except for `Amazon` on which the average query cost is 23.39 (*EG-IGS-E*) vs. 20.05 (BinG) and 21.87 (STBIS), and *EG-IGS-H* outperforms all competitors on `ImageNet`, `Coco` and `NCBIV`.

**Study 4: Impact of Example Quantity.** In this experiment, we vary a percentage $\alpha \in \{0\%, 10\%, 20\%, 50\%, 80\%, 100\%\}$ of nodes in $H$ that have examples. For each value of $\alpha$, we randomly keep the external examples of $\alpha$ portion of nodes at each level in $H$. In particular, with $\alpha = 0\%$, *EG-IGS-E* degenerates to *EG-IGS-H*.

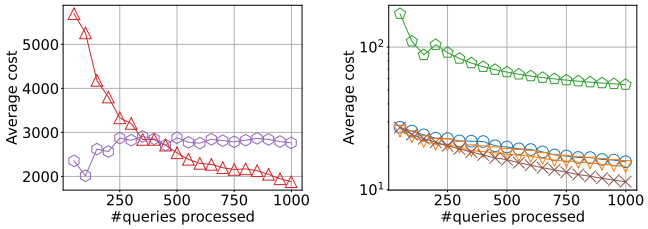
Figure 7a shows the average query costs of *EG-IGS-E*. As expected, more external examples lead to less query cost of *EG-IGS-E* , confirming the importance of guidance from examples. This impact is stronger on those datasets with nodes having large out-degrees, e.g., `NCBI`. Also, with only 10% of nodes having examples, *EG-IGS-E* already outperforms all competitors on all datasets, reducing the average query cost by at least 8.91% (2520.87 vs. 2745.60 for BinG on `NCBI`) and up to 549.26% (1181.26 vs. 7669.43 for MIGS on `NCBI`).

**Study 5: Impact of Example Quality.** We vary the quality of the external examples as follows. For image and text entities, after computing their embeddings $\vec{v}$ which are used for similarity calculation, we replace $\vec{v}$ with an embedding $\vec{u}$ that has $\delta$ cosine distance away from $\vec{v}$. To generate such




(a) Impact of quantity (b) Impact of quality

Fig. 7: Impact of external example entities




(a) On `YAGO3` and `NCBI` (b) On the other datasets

Fig. 8: Average cost vs. #entities processed (*EG-IGS-H*)

an embedding $\vec{u}$, we randomly pick two dimensions $x$, $y$ and rotate $\vec{u}$ from $\vec{v}$ clockwise on the $(x, y)$-plane until the cosine distance between $\vec{u}$ and $\vec{v}$ becomes a specified distance parameter $\delta$. Meanwhile, we constrain that the norms of $\vec{u}$ and $\vec{v}$ should be the same (namely, $v_x^2 + v_y^2 = u_x^2 + u_y^2$). If there is no solution for the selected $x$, $y$, we randomly select another dimension pair until it sorts out. For gene sequence entities, given the original sequence $s_o$, we randomly replace $\delta \cdot \text{length}(s_o)$ characters with different characters randomly picked in $\{A, T, G, C\}$. Thus, the resultant sequence will have $\delta$ edit distance from the original one.

Figure 7b shows the results with $\delta \in \{0, 0.01, 0.05, 0.1, 0.15, 0.2\}$. The embedding quality of the external examples has a mild impact on the performance of *EG-IGS-E*. This may be because our default similarity threshold value, $\tau = 0.6$, is quite small already. As a result, most of the examples similar to $\varphi$ can still be captured, even with the altered embeddings. At $\delta = 0.2$, *EG-IGS-E* outperforms all competitors, by at least 3.04% (25.69 vs. 26.47 for STBIS on `Coco`) and up to 491 times (15.57 vs. 7669.43 for MIGS on `YAGO3`).

**Study 6: Impact of Cold Start.** Next, we look into the impact of *cold start* to our *EG-IGS-H*. We plot the **running average query cost** calculated as query entities are processed by *EG-*

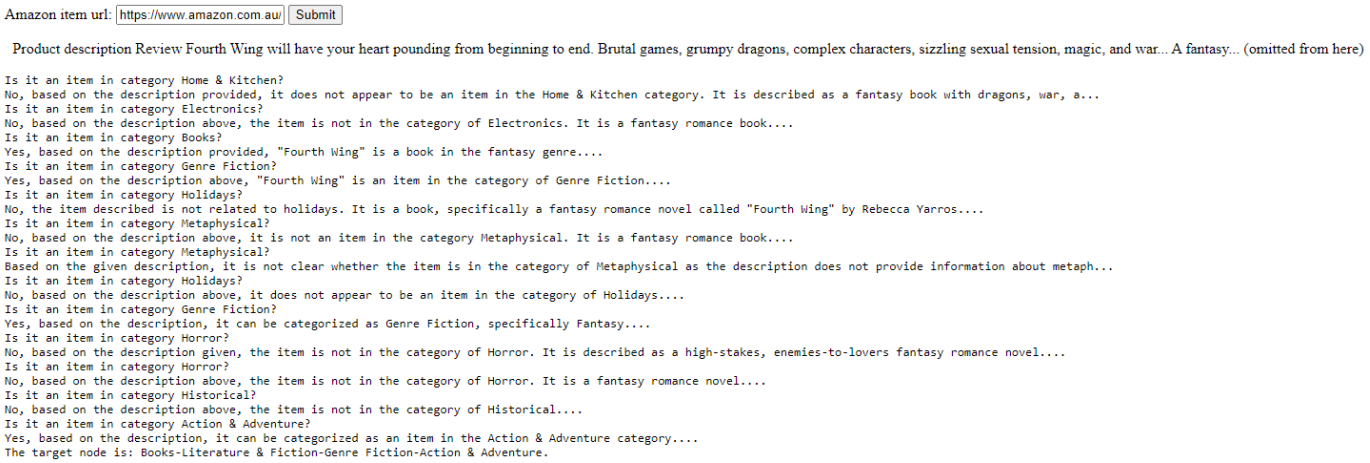



Fig. 9: A screenshot of our ProCat output result

*IGS-H*. Figure 8 shows the results. We find that for most datasets except for NCBI, at the start, the average query costs fluctuate when no good examples are available. However, the query cost steadily reduces when more queries are processed. This confirms the intuition that using examples helps reduce the query costs of our algorithms. After about 250 queries are processed, the running average costs start converging to the overall average cost. The trend is not obvious for NCBI. This is because, in NCBI, each query entity has a distinct target concept. As such, the previous queries have limited benefit.

### E. *Efficiency of Our* Compress-and-Find

The last experiment is to study the efficiency of our *Compress-and-Find* (*CaF*) algorithm. As discussed in Section IV, the query cost complexity of *CaF* is $O(\log h)$, which is crucial to the establishment of the optimality of our *EG-IGS*. We compare *CaF* with the naive algorithm and *TS-IGS* searching on the promising question tree $T_Q$ only. As shown in Table V, the average query cost of *CaF* is consistently smaller than those of the other algorithms on all the datasets. Especially, *CaF* outperforms the naive algorithm by almost an order of magnitude and outperforms *TS-IGS* by up to 3.5 times. This is consistent with their query cost complexity on $T_Q$, where that of *TS-IGS* is $O(\log^2 h)$ and that of the naive algorithm is $O(h)$.

TABLE V: Average Searching Cost on $T_Q$

| | ImageNet | Coco | Amazon | YAGO3 | NCBI | NCBIV |
|---|---|---|---|---|---|---|
| Naive | 44.88 | 24.09 | 14.67 | 16.04 | 19.18 | 10.77 |
| *TS-IGS* | 9.56 | 5.31 | 11.72 | 8.60 | 9.29 | 6.48 |
| *CaF* | 4.96 | 3.71 | 6.93 | 7.25 | 7.09 | 1.84 |

## VII. A Fully-Automatic IGS Demo System

To demonstrate the true practicality of our *EG-IGS* algorithm, we developed a *fully-automatic* IGS system, called ProCat, for Amazon product categorization, with GPT-3.5 [12] serving as an oracle.

**Concept Hierarchy.** ProCat adopts the hierarchy $H$ derived from the Amazon 2014 meta-dataset [29] as introduced in Section VI.

**Example Preparation.** To initialize the system, we pick 400 concepts at leaf nodes in $H$ uniformly at random and then search for items on the Amazon website with these concepts as keywords, taking one item per concept. Next, we crawl the *product description* $P$ of each item on Amazon, by invoking a sub-routine called *get-list-of-words*, which (1) removes from $P$ all the stop words in the `stopwords` list computed by `nltk.corpus` and (2) performs stemming for the remaining words with the `PorterStemmer` in `nltk.stem` library. Denote the resultant list of words by $W_P$. We take $W_P$ as an example for the corresponding concept in $H$. We did not pick all the concepts in $H$ to find examples as 400 examples already work well for our purpose and considerably save time for finding similar examples when processing queries in ProCat.

**Similarity Measurement.** Given a query entity $\varphi$ (i.e., a product description in this context) and a word list $W_P$, we first compute the word list $W_\varphi \leftarrow$ *get-list-of-words* $(\varphi)$, and then compute a similarity score between $W_\varphi$ and $W_P$ with *spaCy* [32]. This score is normalised into a real number in $[0, 1]$ and returned as the desired similarity.

**Questions to GPT-3.5.** Given a query entity $\varphi$ (i.e., product description) and a concept $t \in H$, the question $q(\varphi, t)$ asked to the oracle GPT-3.5 in ProCat is in the following form:

> Based on the following product description, [product description $\varphi$], is this item in category [concept $t$]?

In general, the answer from GPT-3.5 starts with a word either YES or NO. In this case, ProCat just checks the first word from the answer. Otherwise, ProCat invokes the `SentimentIntensityAnalyzer` of the `nltk.sentiment` library for sentiment test. If the sentiment is positive, then it considers the answer as YES; otherwise, NO.

**The ProCat System.** ProCat takes a URL of an item on the Amazon website as the input and works as simply as follows:

1) crawl the product description $\varphi$ from the input URL;
2) invoke our *EG-IGS* for $\varphi$ with all the above components;

The source code and a demon video can be found at this link[5].

[5]https://anonymous.4open.science/r/EG-IGS-AND-GPT-DEMO-C5BD

A screenshot of our ProCat is shown in Figure 9. The top-left form is for URL input, followed by the crawled product description. Due to space limitations, the complete description isn't shown in the screenshot. Below the form, you can find the questions we sent to GPT-3.5 and the corresponding answers presented line by line. Finally, the identified target node is shown on the last line.

## VIII. Conclusion

In this paper, we studied the *interactive graph search* (IGS) problem. **In theory aspect**, we proposed the *Target-Sensitive IGS* (*TS-IGS*) algorithm that achieves a query cost complexity $O(\log n \cdot \log \frac{L}{\log n} + d \cdot \log_d n)$ improving the state-of-the-art bound and matching the theoretical lower bound when $L \in O(\log n)$, where $L$ is the length of the path from the root to the target concept. **In practicality aspect**, we proposed the *Example-Guided IGS* (*EG-IGS*) algorithm, which exploits the knowledge of entities and asks promising questions guided by examples similar to the query entity. *EG-IGS* not only achieves a finer-grained query cost bound than our *TS-IGS*, but also is extremely efficient in practice. Extensive experiments on six real-world datasets involving different types of data show that our *EG-IGS* outperforms all the existing competitors by up to two orders of magnitude in terms of query cost. To further demonstrate the true feasibility of our *EG-IGS* technique, we also developed a fully-automatic Amazon product categorization demo system with GPT-3.5 serving as an oracle.

## References


[1] Y. Tao, Y. Li, and G. Li, "Interactive graph search," in *SIGMOD*, 2019, pp. 1393–1410.

[2] P. Smyth, U. Fayyad, M. Burl, P. Perona, and P. Baldi, "Inferring ground truth from subjective labelling of venus images," in *NIPS*, 1994, pp. 1085–1092.

[3] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones," in *MobiSys*, 2010, pp. 77–90.

[4] P. Welinder and P. Perona, "Online crowdsourcing: Rating annotators and obtaining cost-effective labels," in *CVPR*, 2010, pp. 25–32.

[5] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *PVLDB*, vol. 5, no. 11, pp. 1483–1494, 2012.

[6] N. Vesdapunt, K. Bellare, and N. Dalvi, "Crowdsourcing algorithms for entity resolution," *PVLDB*, vol. 7, no. 12, pp. 1071–1082, 2014.

[7] S. Wang, X. Xiao, and C.-H. Lee, "Crowd-based deduplication: An adaptive approach," in *SIGMOD*, 2015, pp. 1263–1277.

[8] E. Hovy and J. Lavid, "Towards a 'science' of corpus annotation: a new methodological challenge for corpus linguistics," *International Journal of Translation*, vol. 22, no. 1, pp. 13–36, 2010.

[9] M. Sabou, K. Bontcheva, L. Derczynski, and A. Scharl, "Corpus annotation through crowdsourcing: Towards best practice guidelines." in *LREC*, 2014, pp. 859–866.

[10] I. Mozetič, M. Grčar, and J. Smailović, "Multilingual twitter sentiment classification: The role of human annotators," *PLOS ONE*, vol. 11, no. 5, p. e0155036, 2016.

[11] Amazon, "Amazon mechanical turk," June 2023. [Online]. Available: https://www.mturk.com/

[12] OpenAI, "Gpt-3.5," June 2023. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[13] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM*, vol. 32, no. 3, pp. 652–686, 1985.

[14] J. L. Bentley and A. C.-C. Yao, "An almost optimal algorithm for unbounded searching," *Information Processing Letters*, vol. 5, no. SLAC-PUB-1679, 1976.

[15] tfhub.dev, "Tensorflow hub," June 2023. [Online]. Available: https://tfhub.dev/tensorflow/efficientnet/lite0/feature-vector/2

[16] Y. Li, X. Wu, Y. Jin, J. Li, and G. Li, "Efficient algorithms for crowd-aided categorization," *PVLDB*, vol. 13, no. 8, pp. 1221–1233, 2020.

[17] X. Zhu, X. Huang, B. Choi, J. Jiang, Z. Zou, and J. Xu, "Budget constrained interactive search for multiple targets," *PVLDB*, vol. 14, no. 6, pp. 890–902, 2021.

[18] Q. Cong, J. Tang, Y. Huang, L. Chen, and Y. M. Chee, "Cost-effective algorithms for average-case interactive graph search," in *ICDE*, 2022, pp. 1152–1165.

[19] Q. Cong, J. Tang, K. Han, Y. Huang, L. Chen, and Y. M. Chee, "Noisy interactive graph search," in *KDD*, 2022, pp. 231–240.

[20] J. M. Bernardo and A. F. M. Smith, *Bayesian theory*. John Wiley & Sons, 2009, vol. 405.

[21] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Human-assisted graph search: It's okay to ask questions," *PVLDB*, vol. 4, no. 5, p. 267–278, 2011.

[22] W. Kong and J. Allan, "Extending faceted search to the general web," in *CIKM*, 2014, pp. 839–848.

[23] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania, "Minimum-effort driven dynamic faceted search in structured databases," in *CIKM*, 2008, pp. 13–22.

[24] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das, "Facetedpedia: Dynamic generation of query-dependent faceted interfaces for wikipedia," in *WWW*, 2010, pp. 651–660.

[25] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma, "Discovering all most specific sentences," *ACM Transactions on Database Systems*, vol. 28, no. 2, pp. 140–174, 2003.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.

[27] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014, pp. 740–755.

[29] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *WWW*, 2016, pp. 507–517.

[30] F. Mahdisoltani, J. Biega, and F. Suchanek, "Yago3: A knowledge base from multilingual wikipedias," in *CIDR*, 2014.

[31] N. C. f. B. I. Bethesda (MD): National Library of Medicine (US), "National center for biotechnology information," August 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/

[32] spacy.io, "spacy," June 2022. [Online]. Available: https://spacy.io/