Proceedings of the 4th International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/83/2025.20191

A Study on Theoretical Modified Interactive Graph Search Algorithm

Xiuqi Zhu[1,a,*]

[1]*Shanghai Jiao Tong University High School IB Programme, Shanghai, China*
a. 2621880810@qq.com
**corresponding author*

Abstract: Interactive search on hierarchical structures, such as trees and Directed Acyclic Graphs (DAGs), is a crucial problem with applications in various domains, including data retrieval, recommendation systems, and biological networks. Existing methods for interactive search exhibit significant limitations in both theoretical and practical aspects, particularly in handling high-degree nodes and minimizing the number of queries. In this paper, we present a novel approach to interactive graph search that reduces the gap between the current performance bounds and the theoretical optimum. We introduce new algorithms, including an improved *Golden Search* for binary trees and a method for *Equivalence Tree Rewrite*, that efficiently manage high-degree nodes and enhance the overall retrieval performance. Our theoretical analysis establishes tighter lower bounds for the number of queries, and our experimental results on multiple real-world datasets demonstrate significant improvements over state-of-the-art methods. The proposed approach not only achieves better performance in terms of query reduction but also provides a more robust framework for practical applications in complex hierarchical datasets.

Keywords: Interactive Graph Search, Hierarchical Graph, Binary Search, Fibonacci Tree.

1. Introduction

Interactive search is a process where users interact with a retrieval system, refining their queries to obtain more accurate search results. During this process, hierarchical structures serve as a valuable tool to aid in retrieval. Hierarchical structures are ubiquitous in daily life, found in biological classification, workflow diagrams, directory searches, and more. Typically, these structures can be divided into tree structures and Directed Acyclic Graphs (DAGs). A tree structure is a specific type of hierarchical graph where each node has only one parent. In contrast, in a DAG, some nodes may have multiple parents. For example, in the animal classification diagram (Figure 1), a "whale" has two parent categories: "mammal" and "aquatic animal."

© 2025 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/83/2025.20191

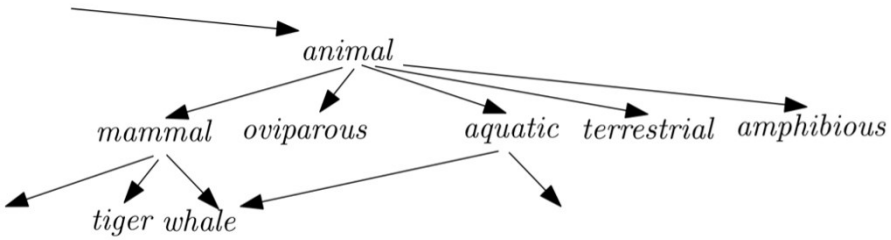

Figure 1: An example of hierarchical graph.

The goal of interactive search on hierarchical structures is to achieve accurate retrieval results with as few interactions as possible. For instance, when searching for a "tiger" in an animal classification diagram, one might ask, "Is the target an aquatic animal?" and receive a negative response. Subsequently, asking, "Is the target a mammal?" and receiving a positive response would help narrow down the search. Through a series of such interactive queries, one can precisely locate the search target.

Interactive search on hierarchical structures has widespread applications in everyday life. For example, when adding a new node to a tree structure, interactive search can help quickly pinpoint the correct position for the new node. Similarly, when identifying errors in a workflow diagram, interactive search can be used to locate the problematic step. In some software applications, interactive search algorithms can be employed to categorize user preferences and provide personalized recommendations for new users. Therefore, efficient interactive search algorithms are crucial for practical applications.

The current methods for solving the problem of interactive search in graph-based data structures exhibit several notable limitations from both theoretical and practical perspectives. These limitations are discussed as follows:

- **Theoretical Limitations:** From a theoretical perspective, the existing methods have a lower bound on the number of interactive queries required, defined as:

$$O\left(\log_2 n \times \left(\log_2 h + d\right)\right)$$

Where n represents the number of nodes, h denotes the depth of the tree-like structure, and d is the maximum degree of the nodes. This theoretical lower bound is significantly higher than the theoretical optimal lower bound of:

$$O\left(\left(d-1\right) \times \log_d n\right)$$

This disparity indicates a substantial gap between the performance of existing methods and the theoretical optimum. The gap suggests that there is considerable room for improvement in reducing the number of interactive queries required to achieve efficient retrieval.

- **Effective Degradation with High-Degree Nodes:** In datasets where nodes have high degrees, i.e., when d is large, the efficiency of current retrieval methods diminishes considerably. This is a critical issue because such high-degree nodes are common in real-world datasets, particularly in social networks, biological networks, and other complex graph structures. The existing methods do not scale well with increasing node degrees, leading to longer retrieval times and higher computational costs, which hampers their practical applicability in scenarios involving large datasets with numerous high-degree nodes.

DOI: 10.54254/2753-8818/83/2025.20191

Despite existing literature providing theoretical analysis of the number of interactive queries required by current methods compared to optimal algorithms, there remains a significant gap between the current methods and the theoretical optimum. Therefore, this project aims to further analyze the problem from a theoretical perspective. By doing so, we seek to tighten the theoretical lower bounds of current methods and explore avenues to develop a theoretically optimal solution. This involves revisiting the underlying assumptions and constraints of the problem and devising new algorithms that can achieve a closer approximation to the theoretical optimum.

From an application standpoint, when all nodes or the child nodes of a particular parent node are numerous, the efficiency of current search methods declines sharply. According to relevant literature, this situation is commonly encountered in real-world datasets. To address this, our project will analyze and improve upon existing algorithms to enhance performance under conditions where node degrees are excessively large. This involves developing new strategies to manage high-degree nodes more effectively, potentially through heuristic-based optimizations or dynamic data structuring techniques. By doing so, we aim to achieve significant improvements in retrieval efficiency when applying these methods to real-world datasets.

The key contributions of the paper are summarized as follows:

- **Improved Algorithm for Interactive Search:** The paper proposes an enhanced algorithm for conducting interactive searches in hierarchical data structures, particularly in DAGs and tree structures. The primary objective of the proposed algorithm is to minimize the number of interactive queries required to accurately identify a target node within a given structure.
- **Theoretical Analysis and Optimality:** From a theoretical perspective, the paper revisits the problem of interactive search and establishes tighter lower bounds on the number of queries required. The existing methods have a lower bound of:

$$O\left(\log_2 n \times \left(\log_2 h + d\right)\right)$$

Where n is the number of nodes, h denotes the depth of the tree-like structure, and d represents the maximum degree of the nodes. However, the theoretical optimal lower bound is $O\left((d-1) \times \log_d n\right)$.The paper identifies a substantial gap as $O\left(\log_\phi n\right)$, where n is the number of nodes, $\phi = \frac{\sqrt{5}+1}{2}$. between these bounds and aims to develop algorithms that achieve performance closer to the theoretical optimum.

- **Handling High-Degree Nodes:** The paper addresses the performance degradation of current methods in the presence of high-degree nodes, which are common in real-world datasets, such as social networks and biological networks. The proposed algorithm introduces strategies for effectively managing nodes with large degrees to improve the overall efficiency of the search process. This is achieved through heuristic-based optimizations and dynamic data structuring techniques.
- **Equivalence Tree Rewrite and Golden Search:** The paper introduces the concept of *Equivalence Tree Rewrite*, which transforms a given hierarchical tree into a new structure that simplifies the search process while preserving the original tree's properties. Additionally, the paper proposes the *Golden Search* algorithm for efficiently locating a target node within a left-heavy binary tree, reducing the number of queries needed by leveraging the properties of the tree structure.
- **Extensive Experimental Evaluation:** The proposed methods are extensively evaluated on multiple real-world datasets, including Amazon, ImageNet, ACM CCS, and ProdClass. The experimental results demonstrate that the proposed algorithms significantly outperform existing methods, such as IGS, TS-IGS, BinG, and KBM-IGS, in terms of reducing the average number of

DOI: 10.54254/2753-8818/83/2025.20191

queries required to find a target. For instance, the proposed method requires only *11.82* queries on average in the ProdClass dataset, compared to *34.80* queries for the baseline IGS method.

2. Related Work

We begin by summarizing and comparing four key studies closely related to our work: IGS [1], TS-IGS [2], BinG [3,4], and KBM-IGS [5]. Tao et al. [1] introduced the Interactive Graph Search (IGS) problem, which dynamically adapted to feedback to enhance target identification within Directed Acyclic Graphs (DAGs). The core approach involved extracting a heavy DFS-tree and applying binary search along with heavy-path decomposition. Building on IGS, Zhao et al. [2] proposed the TS-IGS algorithm, which improved the efficiency of the search by reducing the number of questions required when the target was located at shallower depths in the graph. BinG [3,4] presented a greedy-based method that prioritized querying the vertex with maximal information gain, modeling the single-target problem as a decision tree construction task, and provided theoretical guarantees for its approximation performance. Extending the problem to a multi-target setting, Zhu et al. [5] introduced the KBM-IGS framework, which was the first to address multiple targets within tree structures. This heuristic method balanced target probability and benefit gain while incorporating budget constraints to effectively approximate the target set under limited query budgets.

Several additional graph search problems have also been explored in relation to our work [3,4,6,7,9,10,11,12,13]. Human-assisted graph search [6] focuses on optimizing the cost of human computations in crowdsourcing services. This offline problem requires all questions to be asked in one batch. Cong et al. [4] addressed the minimization of the expected number of questions, under the assumption that objects follow a probabilistic distribution, though acquiring this distribution accurately in practice remains challenging. Li et al. [3,7] proposed a multiple interactive graph search, incorporating multiple-choice questions to improve the querying process. In another work [8], the authors examined the impact of human errors on the search process within the IGS framework, considering the effect of inaccurate responses. Lu et al. [9-11] investigated the partial-order multiway search problem, where k questions are posed in each round. In the case of a single question per round, their approach aligns with the IGS method [1].

3. Preliminaries

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

3.1. Hierarchical DAG

Let $H = (V, E)$ be hierarchical DAG with a set V of vertices and a set E of directed edges. The edges of H have directions, but H has no directed cycles. For an edge $\langle u, v\rangle$, we say u is an in-neighbor of v and v is an out-neighbor of u. For a vertex v in H, we denote the set of in-neighbors of v by $N^+(v) = \{u: \langle u, v\rangle \in E\}$ and the set of out-neighbors of v by $N^-(v) = \{u: \langle v, u\rangle \in E\}$. We consider that H has only one root r. If this is not true, simply add a virtual vertex to H with an outgoing edge to every vertex $N^+(v) = \emptyset$. For any two vertices u and v, we say that u can reach v (denoted as $u \to v$), if and only if there exists a directed path from u to v in H. If u can not reach v, it is denoted as $u \nrightarrow v$. In addition, the ancestors of a vertex v, denoted by $anc(v)$, are defined to be the set of vertices that can reach v in H, i.e., $anc(v) = \{u \in V: u \to v\}$. Similarly, the descendants of a vertex v, denoted by $des(v)$, are defined to be the set of vertices that are reachable from v in H, i.e., $des(v) = \{u \in V: v \to u\}$.

DOI: 10.54254/2753-8818/83/2025.20191

3.2. Binary-or Questions and Problem Definition

Binary questions. Give a target vertex $t \in V$, an algorithm is allowed to ask a binary question $reach(q)$, where $q \in V$. The boolean answer of $reach(q)$ is either Yes or No. If $reach(q) = Yes$, $q \to t$; Otherwise, if $reach(q) = No$, $q \nrightarrow t$.

Binary-or questions. Different from the binary questions, binary-or questions allow an algorithm to ask questions with a set of vertex $reach(S)$. Here, the vertices in S should be the subclass of the same object, i.e., $\exists v \in V$ such that $S \subseteq N^-(v)$. The boolean answer of $reach(S)$ is also either Yes or No. If $reach(S) = Yes$, $\exists q \in S$ such that $q \to t$; Otherwise, if $reach(S) = No$, $\forall q \in S$, $q \nrightarrow t$.

Definition 1 (Interactive Graph Search with Binary-Or Questions). *Given a hierarchical DAG $H = (V, E)$ and an unknown target $t \in V$, the object of an algorithm is to ask minimal binary questions to identify the unknown target t.*

3.3. Problem Analysis

Candidates. In the IGS problem, we define all the nodes that may be potential targets as candidates, denoted as $\mathcal{C}$. For the initialization, the candidates are all the nodes in the HDAG.

Rule 1 (Candidate Update). *After a question $reach(q)$, if the answer is Yes, the candidates $\mathcal{C}_{yes} = \mathcal{C} \cap des(q)$. If the answer is No, the candidates $\mathcal{C}_{no} = \mathcal{C} \backslash des(q)$.*

Rule 2 (Find Exact Target). *If there only exists one nodes in the candidates, i.e., $|\mathcal{C}| = 1$, we could find the exact target t that $t \in \mathcal{C}$.*

Gain. The gain of a question is the reduction of candidates size, i.e., for the yes gain, $G_{yes} = |\mathcal{C}| - |\mathcal{C}_{yes}|$, and for the no gain $G_{no} = |\mathcal{C}| - |\mathcal{C}_{no}|$.

4. Solution

4.1. Chain

Binary Search on Chain. First, we discuss a special case chain. In the special case, we could apply binary search on the chain case. Next, we could prove the upper bound and lower bound on the chain dataset.

Theorem 1 (Upper Bound). *Given a chain T, binary search asks at most $log_2 n$ questions in the worst case. Here n is the number of nodes.*

Proof. On the chain T, binary search can find a node x that $des(x) = \lfloor \frac{n}{2} \rfloor$. So, after the question $reach(x)$, the rest size of candidate is at most $\lceil \frac{n}{2} \rceil$. So, binary search asks at most $log_2 n$ questions. □

Theorem 2 (Lower Bound). *Given a chain T, any algorithm asks at least $log_2 n$ questions in the worst case. Here n is the number of nodes.*

Proof. For any question x, if $des(x) < \lceil \frac{n}{2} \rceil$, we get No answer and the rest size of candidate is at least $\lceil \frac{n}{2} \rceil$. Otherwise, we get Yes answer, the rest size of candidate is also larger than $\lceil \frac{n}{2} \rceil$. So, after a question, in the worst case, the rest size of candidate is larger than $\lceil \frac{n}{2} \rceil$. So, any algorithm asks at least $log_2 n$ questions. □

4.2. Binary Tree

4.2.1. Binary Search

Algorithm 1 Binary Search on Tree

Require: A hierarchical tree $T = (V, E)$.

DOI: 10.54254/2753-8818/83/2025.20191

Ensure: The target node t.
1: Let $\mathcal{C} \leftarrow V$, $t \leftarrow root$;
2: **While** $|\mathcal{C}| > 1$ **do**
3: Calculate G_{yes} and G_{no};
4: **for** $v \in \mathcal{C}$ **do**
5: $Gain(v) = min(G_{yes}, G_{no})$;
6: $q \leftarrow \text{argmax}_{v \in \mathcal{C}} Gain(v)$;
7: Ask the question $reach(q)$;
8: **if** $reach(q) =$ Yes **then**
9: $C = C \cap des(q)$;
10: $t = q$;
11: else
12: $C = C \backslash des(q)$;
13: **return** t

Binary Search on Tree. Algorithm 1 presents a "Binary Search on Tree" method designed to efficiently locate a target node t within a hierarchical tree $T = (V, E)$. The algorithm operates by iteratively narrowing down the candidate set of nodes $\mathcal{C}$ using a binary search approach adapted for a tree structure.

Initially, the candidate set $\mathcal{C}$ is set to include all nodes V in the tree, and the target node t is initialized to the root of the tree. The algorithm proceeds with a loop that continues as long as there is more than one candidate node in $\mathcal{C}$. During each iteration, the algorithm computes two metrics: G_{yes} and G_{no}, which represent the potential gains (or benefits) of answering "yes" or "no" to a query about the reachability of a particular node.

For each node v in the candidate set $\mathcal{C}$, the algorithm calculates the gain $Gain(v) = \min(G_{yes}, G_{no})$. This gain represents the minimal reduction in the size of the candidate set, helping to identify the most informative query. The algorithm then selects the node q that maximizes the gain, i.e., $q = \text{argmax}_{v \in \mathcal{C}} Gain(v)$, and poses the interactive question $reach(q)$.

If the query $reach(q)$ returns yes, indicating that the target node t is reachable from q, the candidate set $\mathcal{C}$ is updated to include only the descendants of q, and the target node t is updated to q. If the query returns no, the candidate set $\mathcal{C}$ is pruned to exclude the descendants of q.

This iterative process continues, systematically reducing the size of the candidate set $\mathcal{C}$ until only one node remains, which is returned as the target node t. The algorithm leverages the hierarchical structure of the tree to efficiently converge to the target, ensuring that the search is conducted with minimal queries and maximum effectiveness.

Theorem 3 (Upper Bound of Binary Search). The upper bound of binary search on binary tree is $O\left(log_{\frac{3}{2}} n\right)$ [14].

DOI: 10.54254/2753-8818/83/2025.20191

4.2.2. Lower Bound of Binary Tree

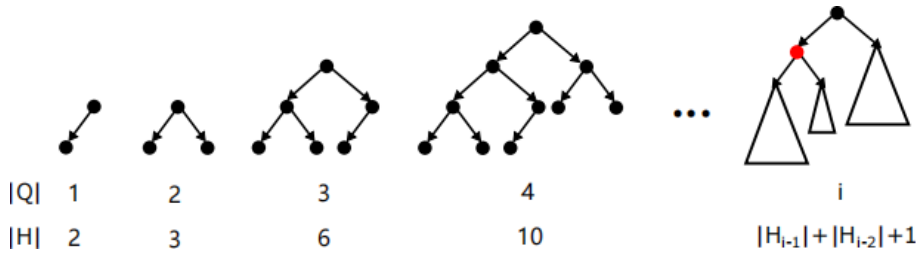

Figure 2: Worst Case

Definition 2 (Fibonacci Tree). For the i-th Fibonacci Tree T_i, the left son of root is the tree T_{i-1} and the right son is T_{i-2}.

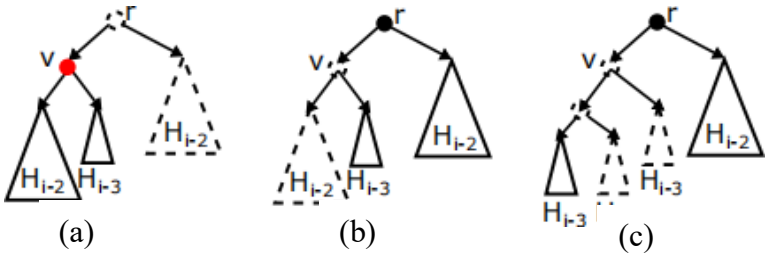

Figure 3: Proof of Worst Case

Lemma 1 (Fibonacci Tree Depth). In the Fibonacci Tree, the number of node:

$$n = \left(\frac{1+\sqrt{5}}{2}\right)^h + \left(\frac{1-\sqrt{5}}{2}\right)^h - 1.$$

Here, h is the depth of the Fibonacci Tree, $h = \log_\phi n$.

Lemma 2. Given a hierarchical tree H, for any subtree $H' \subset H$, $Q(H') \leq Q(H)$.

Theorem 4 (Lower Bound on Fibonacci Tree). Any algorithm should ask at least $\log_\phi n$ questions on Fibonacci Tree. Here, n is the number of nodes, $\phi = \frac{\sqrt{5}+1}{2}$.

Proof. We first assume the Fibonacci Tree $Q(H_{i-1}) = i - 1$. Then, consider the Fibonacci Tree H_i,

- If the algorithm ask the node v, then it gets the yes answer. So, the rest subtree is the H_{i-1} shown in Figure 3(a), which need ask at least $Q\left(H_{i-1}\right) = i - 1$ question.
- If the algorithm ask the right child of node r, then it gets the No answer.So, the rest subtree T shown in Figure 3(a), where H_{i-1} is the subtree of T. So it need ask at least $Q\left(H_{i-1}\right) = i - 1$ question.
- If the algorithm ask the left child node of v, then it gets the No answer. So, the rest subtree T shown in Figure 3(b), where H_{i-1} is the subtree of T. So it need ask at least $Q\left(H_{i-1}\right) = i - 1$ question.

DOI: 10.54254/2753-8818/83/2025.20191

- If the algorithm ask the right child of node v, then it gets the No answer.So, the rest subtree T shown in Figure 3(c), where H_{i-1} is the subtree of T. So it need ask at least $Q\left(H_{i-1}\right) = i - 1$ question.

  By induction, we can proof for any Fibonacci Tree H_i:

$$Q(H_i) = i = \log_{\phi} n.$$

4.2.3. Golden Search

Algorithm 2 Golden Search

Require: A left heavy binary tree $H = (V, E)$ rooted at r.
Ensure: A identified target t^*.
Let $\mathcal{C} \leftarrow V$, $t^* \leftarrow r$;
While $|\mathcal{C}| > 1$ **do**
  for $v \in \mathcal{C}$ **do**
    $g_1 \leftarrow \frac{|\mathcal{C} \backslash des(v)|}{|\mathcal{C}|}$;
    $g_2 \leftarrow \max\{\frac{|\mathcal{C} \cap des(lc(v))|}{|\mathcal{C}|}, \frac{|\mathcal{C} \cap des(rc(v))|+1}{|\mathcal{C}|}\}$;
    $g(v) \leftarrow \max\{g_1, \sqrt{g_2}\}$;
  $q_i \leftarrow \operatorname{argmin}_{v \in \mathcal{C}} g(v)$;
  Ask the question $reach(q_i)$;
  if $reach(q_i) =$ Yes **then**
    Ask the question $reach\left(lc(q_i)\right)$;
    if $reach\left(lc(q_i)\right) =$ Yes **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cap des\left(lc(q_i)\right), t^* \leftarrow lc(q_i)$;
    else
      $\mathcal{C} \leftarrow \mathcal{C} \cap des\left(lc(q_i)\right), t^* \leftarrow lc(q_i)$;
  else
    $\mathcal{C} \leftarrow \mathcal{C} \backslash des(q_i)$;
return t^*

The "Golden Search" algorithm, outlined in Algorithm 2, is designed to efficiently identify a target node t^* within a left-heavy binary tree $H = (V, E)$ rooted at r. The algorithm operates iteratively, leveraging properties of the binary tree structure to minimize the number of queries needed to locate the target.

The algorithm begins by initializing the candidate set $\mathcal{C}$ to all nodes V and setting the initial target estimate t^* to the root node r. The main loop continues as long as the candidate set $\mathcal{C}$ contains more than one node. For each node v in the candidate set, the algorithm computes two parameters, g_1 and g_2, which are ratios representing the relative sizes of various subsets of $\mathcal{C}$. The value g_1 is the proportion of nodes in $\mathcal{C}$ that are not descendants of v, while g_2 is the maximum of two quantities: the fraction of nodes in $\mathcal{C}$ that are descendants of the left child of v, and the fraction of nodes that are either descendants of the right child or the node itself, incremented by one. The goal is to balance the search between these subsets.

The algorithm then defines $g(v)$ as the maximum of g_1 and the square root of g_2, which represents a heuristic measure for selecting the next query. The node q_i that minimizes $g(v)$ is

DOI: 10.54254/2753-8818/83/2025.20191

chosen for the next query $reach(q_i)$. If the query returns yes, indicating that the target is reachable from q_i, a subsequent query is made to $reach(lc(q_i))$ to determine if the target is in the left subtree. Based on the responses to these queries, the candidate set $\mathcal{C}$ is updated accordingly to focus the search on the subtree containing the target. If the target is not reachable from q_i, the candidate set is pruned to exclude the descendants of q_i.

This process iteratively narrows down the candidate set until only one node remains, which is returned as the identified target t^*. The algorithm ensures that the search is conducted efficiently, minimizing the number of queries required to locate the target in the left-heavy binary tree.

Lemma 3. *Given a hierarchical binary tree H, there exists a node x, satisfied 1) $sz(x) \geq a$. 2) For left child and right child of x, $sz(lc(x)) < a$ and $sz(rc(x)) < a$. Here, n is the number of nodes, $a = n - \frac{n}{\phi} = \frac{n}{\phi^2}$, $\phi = \frac{\sqrt{5}+1}{2}$.*

Proof. The root r satisfies $r \geq a$. For each node x satisfied $x \geq a$, if its left child and right child satisfy the rule $sz(lc(x)) < a$ and $sz(rc(x)) < a$. Then, we find the node x. Otherwise, there exists a child y satisfied $y \geq a$. We can select y. □

Lemma 4. *After each question of x, if $reach(x) = No$, $|\mathcal{C}| \leq \frac{n}{\phi}$. If $reach(x) = Yes$, after two questions, $|\mathcal{C}| \leq \frac{n}{\phi^2}$.*

Proof. If $reach(x) = No$, $|\mathcal{C}| = n - sz(x) \leq n - a = \frac{n}{\phi}$. If $reach(x) = Yes$, we can ask the question of its left child. If it gets yes answer, $|\mathcal{C}| = sz(lc(x)) < a = \frac{n}{\phi^2}$. Otherwise, $|\mathcal{C}| = sz(rc(x)) + 1 \leq a = \frac{n}{\phi^2}$. □

Theorem 5 (Upper Bound for Golden Search). *Given a hierarchical binary tree H, golden search asks at most $\log_\phi n$ questions in the worst case. Here, n is the number of nodes, $\phi = \frac{\sqrt{5}+1}{2}$.*

Proof. Based on the Lemma 3 and 4, we could prove it. □

4.3. Tree and DAGs

Definition 3 (Equivalence Tree Rewrite). Given a hierarchical Tree $G = (V, E)$, a new rewriting hierarchical tree $G' = (V', E')$ is an equivalence tree should satisfies,

1. $V \subseteq V'$.
2. *For any two vertices $u, v \in V$, if u can reach v in the tree G, i.e., $u \to v$, u can reach v in the rewriting tree G'.*
3. *For any two vertices $u, v \in V$, if u can not reach v in the tree G, i.e., $u \nrightarrow v$, u can not reach v in the rewriting tree G'.*

Algorithm 3 Equivalence Tree Rewrite

Require: A hierarchical Tree $G = (V, E)$. .

Ensure: A new hierarchical Tree $G' = (V', E')$.

1: Let $G' \leftarrow G$;
2: **for** $v \in \mathcal{C}$ **do**
3: **while** $N^-(v) > 2$ do
4: Select two nodes x, y in $N^-(v)$;
5: Add a new node v' in G';

DOI: 10.54254/2753-8818/83/2025.20191

6: Add the edges $\langle v, v' \rangle$, $\langle v', x \rangle$, and $\langle v', y \rangle$;
7: Remove the edges $\langle v, x \rangle$ and $\langle v, y \rangle$;
8: **return** G

Equivalence Tree Rewrite. Algorithm 3 presents an "Equivalence Tree Rewrite" method that transforms a given hierarchical tree $G = (V, E)$ into a new hierarchical tree $G' = (V', E')$. The primary objective of this algorithm is to restructure the tree such that each node in the resulting tree has at most two children, thereby simplifying the tree's structure while maintaining its hierarchical properties.

The algorithm begins by initializing the new tree G' to be identical to the input tree G. It then iterates over each node v in the set of vertices V of the tree. For each node v, the algorithm checks if the in-degree (i.e., the number of incoming edges) $N^-(v)$ exceeds two. If so, the algorithm enters a while loop to iteratively reduce the in-degree of v.

Within the loop, two nodes, x and y, are selected from the set of incoming neighbors $N^-(v)$ of node v. A new auxiliary node v' is then added to the tree G'. The algorithm proceeds to add three edges: one connecting v to the newly created node v', and two connecting v' to x and y, respectively. After establishing these new edges, the original edges connecting v to x and v to y are removed from the tree.

This process is repeated until the in-degree of each node v is reduced to at most two. The outer loop ensures that all nodes in the tree are processed in a similar manner. Upon completion of the transformation for all nodes, the algorithm returns the modified tree G'.

Lemma 5. *For each vertex $v \in V$, we add at most $|N^-(v)| - 2$ vertices in Algorithm* 3.

Proof. After adding a new node v', the size of children of v would reduce 1. Due to the size of children should be equal 2, the Algorithm 3 adds at most $|N^-(v)| - 2$ vertices. □

Theorem 6. *Given a hierarchical Tree $G = (V, E)$, where $n = |V|$, the size of equivalence rewriting tree $G' = (V', E')$ is no larger than $2n$, i.e., $|V'| \leq 2n$.*

Proof. On the basis of Lemma 5, for each node v, Algorithm 3 adds at most $|N^-(v)| - 2$ vertices. So, we add at most $\sum_{v \in V} |N^-(v)| - 2 \leq n$ nodes totally. So, the size of equivalence rewriting tree $G' = (V', E')$ is no larger than $2n$. □

Theorem 7. *Given a hierarchical Tree $G = (V, E)$, on the equivalence rewriting tree, the golden search asks at most $O(log_\phi n)$ questions in the worst case. Here, n is the number of nodes, $\phi = \frac{\sqrt{5}+1}{2}$.*

Proof. Based on the Theorem 5 and 6, we could prove it. □

DFS-tree extraction [1]. For the DAG dataset, we could follows the DFS-tree method, similar to [1]. The DFS-Tree algorithm begins at a designated root node and explores as far along each branch as possible before backtracking. This exploration strategy is well-suited for DAGs due to their acyclic nature, ensuring that once a path is followed from a node, there is no need to check for cycles.

5. Experiment

In this section, we conduct extensive experiments to evaluate the performance of our proposed methods. All algorithms are implemented in C++.

We evaluate our proposed algorithms on three real-world datasets, each chosen for its relevance to graph algorithms and unique structural properties.

- **Amazon** [15]: A hierarchical dataset of product categories from Amazon, modeled as a tree. It tracks paths from the root node to specific product categories within Amazon's extensive catalog.

DOI: 10.54254/2753-8818/83/2025.20191

- **ACM_CCS**[1]: The 2012 ACM Computing Classification System, represented as a Directed Acyclic Graph (DAG). This poly-hierarchical ontology is designed to support semantic web applications and enhance search functionalities within the ACM Digital Library.
- **ImageNet** [16]: A large-scale hierarchical image database, organized as a DAG and based on WordNet [17]. It contains a vast array of image categories with detailed ground truth labels, frequently used for image classification tasks.

To provide clarity and avoid redundancy, the specific details regarding vertices, edges, and structures for these datasets are summarized in Table 1.

Table 1: Summary of datasets

Dataset	Vertices($\|V\|$)	Edges($\|E\|$)	Max Degree	Data Domain
Amazon	29,240	29,239	225	Tree
ACM_CCS	1,928	2,113	18	DAG
ImageNet	74,401	75,850	402	DAG

Comparison methods. We compare four state-of-the-art methods for interactive graph search, as outlined below:

- **IGS** [1]: This approach targets the identification of a single vertex within a DAG by first extracting a DFS-tree and then applying heavy-path decomposition to divide the tree into several connected paths. Each path is subsequently searched using binary search to locate the target vertex.
- **TS-IGS** [2]: An extension of IGS, TS-IGS incorporates heavy-path decomposition but introduces a result-sensitive binary search algorithm. This method replaces the standard binary search used in IGS and is particularly effective when the target vertices are located near the shallow regions of the graph G.
- **BinG** [3,4]: A greedy-based algorithm designed for interactive graph search in tree structures. BinG identifies a single target by querying the vertices that most significantly reduce the search space, aiming to minimize the overall graph size.
- **KBM-IGS** [5]: This method extends interactive graph search to multiple-target scenarios within tree structures. With a budget constraint b limiting the number of queries, it employs dynamic programming to select vertices with the highest expected gain, balancing target probability with benefit gain to optimize the search process.

Table 2: The average number of questions for finding the target in IGS problem.

Dataset	IGS	TS-IGS	BinG	KBM-IGS	GoldGS(Ours)
Amazon	30.18	29.15	24.97	24.98	**14.83**
ACM_CCS	18.65	17.96	14.22	14.27	**10.91**
ImageNet	38.60	37.13	31.43	31.71	**16.18**

Experiment result. The results of our experiments, summarized in Table 2, illustrate the effectiveness of our proposed method in reducing the average number of questions required to find the target in the Iterative Graph Search (IGS) problem across multiple datasets. Our approach significantly outperforms existing methods, including IGS, TS-IGS, BinG, and KBM-IGS, in all tested datasets.

[1] https://dl.acm.org/ccs

DOI: 10.54254/2753-8818/83/2025.20191

In the *Amazon* dataset, our method reduces the number of questions to 14.83 compared to 30.18 for IGS, 29.15 for TS-IGS, 24.97 for BinG, and 24.98 for KBM-IGS. The *ACM_CCS* dataset also shows a notable decrease, with our method requiring only 10.91 questions on average, whereas IGS, TS-IGS, BinG, and KBM-IGS need 18.65, 17.96, 14.22, and 14.27 questions, respectively. Lastly, in the *ImageNet* dataset, our approach achieves an average of 16.18 questions, outperforming IGS (38.60), TS-IGS (37.13), BinG (31.43), and KBM-IGS (31.71).

These results demonstrate that our proposed method is more efficient in finding the target across all datasets, confirming its superiority in terms of reducing the average number of questions needed for the IGS problem.

6. Conclusion and Future Work

In this paper, we have presented a comprehensive study on the interactive search of hierarchical structures, focusing on optimizing query efficiency and addressing the limitations of existing methods. Our contributions include the development of the *Golden Search* algorithm for binary trees and the *Equivalence Tree Rewrite* technique, which collectively enhance the search process by reducing the number of interactive queries required to locate a target node. Theoretical analysis has shown that our methods achieve tighter bounds on query complexity, moving closer to the theoretical optimum. Empirical evaluations on various datasets, including Amazon, ImageNet, and ACM CCS, have validated the effectiveness of our approach, demonstrating significant improvements over existing state-of-the-art algorithms.

Despite these advancements, several avenues for future research remain open. One potential direction is to extend our algorithms to handle more general forms of graphs, such as those with cycles or more complex network structures. Additionally, further work could explore the integration of machine learning techniques to predict more effective queries dynamically, thereby improving search efficiency in real-time applications. Finally, applying our methods to broader domains, such as social networks and semantic web applications, could provide further insights and opportunities for optimization. We believe that the approaches introduced in this paper offer a strong foundation for future advancements in interactive graph search and related fields.

Acknowledgments

My recent project of ‘A Study on Theoretical Modified Interactive Graph Search Algorithm’ is a hard job to finish all of the work by my own. I am thankful to my teacher, Mr Yu as a support of my article writing. Also, I am very appreciated by my mother, who gives me inspiration about the project.

When I wrote the article, all the content were in my brain, but I could not organize it very well. Mr Yu, your guidance was nothing short of exemplary. Thanks for helping me structure my thoughts coherently. It’s my first time to write an article before this contest, and your help was essential in conducting meaningful and reliable experiments. I remember that how to organize the structure of the paper in a good logic, how to highlight the important part and how to provide more concrete data to support my study in my paper. Thanks for guiding me to practice my academic writing skills, your dedication to helping me hone my writing skills has been a profound learning experience.

Your expertise and patience in guiding me through the complex process were truly remarkable. You took the time to explain intricate concepts, encouraged me to think critically. Once you said that my solution was not clear enough to represent my train of thought. I tried to use a lot of explain method, and finally the logic becomes clear. Once you said that some formulas inserted in my paper makes people who read the article confused, since it is too chaotic. You suggested me to change the method of showing the formulas, and the passage looks more logical and organized.

DOI: 10.54254/2753-8818/83/2025.20191

I am proudly grateful for your unwavering support and encouragement. Your belief in my abilities has given me the confidence to tackle challenges and strive for excellence. I am truly fortunate to have the opportunity to learn from someone as knowledgeable and dedicated as you. You supported me a lot from structuring my thoughts, teaching me how to use Overleaf to write articles, and finally improving my logics. You gave me these helps in your spare time unrequitedly, and I will never forget these days help.

I really appreciate the help from my mathematic teacher, Mr Yu and the help from my mother. Everything started when my mother taught me about graph theory and the knowledge of trees. My project started when I thought about how to search a point of graph. I asked for my mother's help for some references, and I started to think about how can I improve the methods. I tried to think about how to solve the problems. When I shared my idea of graph search, I got appreciation from my parent and my friends. I tried to improve the effectiveness of the methods in the references, and I found a new method by myself. To justify my hypothesis, I made a code. With my mother's help on getting the sample data and programing my code, I successfully proved that my method has the best effectiveness among all the other algorithms.

I want to pay sincere thanks to all the people who helped me during this project, including my Mathematic Meacher, Mr Yu, my family and my friends.

References

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

[2] *Zhuowei Zhao, Junhao Gan, Jianzhong Qi, and Zhifeng Bao. Efficient example-guided interactive graph search. In 2024 IEEE 40th International Conference on Data Engineering (ICDE), pages 342–354. IEEE, 2024.*

[3] *Yuanbing Li, Xian Wu, YifeiJin, Jian Li, and Guoliang Li. Efficient algorithms for crowd-aided categoriza- tion. PVLDB, 13(8):1221–1233, 2020.*

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

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

[6] *Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: It's okay to ask questions. PVLDB, 4(5):267–278, 2011.*

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

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

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

[10] *Shangqi Lu, Wim Martens, Matthias Niewerth, and Yufei Tao. An optimal algorithm for partial order multi- way search. ACM SIGMOD Record, 52(1):84–92, 2023.*

[11] *Lu Shangqi, Wim Martens, Matthias Niewerth, and Yufei Tao. Partial order multiway search. ACM Transac- tions on Database Systems, 48(4):1–31, 2023.*

[12] *Yingxue Li, Shaohan Chen, and Weiguo Zheng. Knowledge-driven interactive graph search. Beijing Da Xue Xue Bao, 59(5):735–746, 2023.*

[13] *Rajesh Jayaram. Technical perspective: Optimal algorithms for multiway search on partial orders. ACM SIGMOD Record, 52(1):83–83, 2023.*

[14] *Yosi Ben-Asher and Eitan Farchi. The cost of searching in general trees versus complete binary trees. Tech- nical report, Citeseer, 1997.*

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

[16] *Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.*

DOI: 10.54254/2753-8818/83/2025.20191

[17] Christiane Fellbaum. WordNet: An electronic lexical database. MIT press, 1998.