Constrained Binary Identification Problem

Amin Karbasi[1] and Morteza Zadimoghaddam[2]

1 **I&C, EPFL, Switzerland,** amin.karbasi@epfl.ch
2 **CSAIL, MIT, Cambridge-MA-USA,** morteza@mit.edu

Abstract

We consider the problem of building a binary decision tree, to locate an object within a set by way of the least number of membership queries. This problem is equivalent to the "20 questions game" of information theory and is closely related to lossless source compression. If any query is admissible, Huffman coding is optimal with close to H[P] questions on average, the entropy of the prior distribution P over objects. However, in many realistic scenarios, there are constraints on which queries can be asked, and solving the problem optimally is NP-hard.

We provide novel polynomial time approximation algorithms where constraints are defined in terms of "graph", general "cost", and "submodular" functions. In particular, we show that under graph constraints, there exists a constant approximation algorithm for locating the target in the set. We then extend our approach for scenarios where the constraints are defined in terms of general cost functions that depend only on the size of the query and provide an approximation algorithm that can find the target within $O(\log(\log n))$ gap from the cost of the optimum algorithm. Submodular functions come as a natural generalization of cost functions with decreasing marginals. Under submodular set constraints, we devise an approximation algorithm that can find the target within $O(\log n)$ gap from the cost of the optimum algorithm. The proposed algorithms are greedy in a sense that at each step they select a query that most evenly splits the set without violating the underlying constraints. These results can be applied to network tomography, active learning and interactive content search.

1998 ACM Subject Classification G.2.2 Graph Algorithms and Network Problems, I.1.2 Analysis of Approximation Algorithms, H.3.3. Information Search and Retrieval.

Keywords and phrases Network Tomography, Binary Identification Problem, Approximation Algorithms, Graph Algorithms, Tree Search Strategies, Entropy.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Constructing a binary search tree is one of the fundamental problems in discrete mathematics. Formally, we are given a set $\mathcal{N} = \{1, \ldots, n\}$ of objects, identified with integers from 1 to n, as well as a probability distribution $P = (p_1, \ldots, p_n)$ over $\mathcal{N}$, $p_i \geq 0$, $i = 1, \ldots, n$, and $\sum_{i=1}^{n} p_i = 1$. The goal is to locate an object within a set $\mathcal{N}$ by way of the least number of membership queries. Each binary split is a partition of $\mathcal{N}$ into a subset $Q \subset \mathcal{N}$ and its complement $\mathcal{N} \setminus Q$. Given that a membership oracle provides answers without noise, what is the best we can do in a computationally feasible manner? This problem is equivalent to the "twenty questions game" of information theory [7], where a player has to determine the identity of an object from a set (say, a famous person) by asking a minimum number of yes/no questions. If any split is an admissible query, Huffman coding is optimal with close to H[P] questions on average [7], the entropy of the distribution P from which the target object is drawn at random.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, in realistic scenarios, there are *constraints* on which queries can be asked, and solving the problem optimally is NP-hard [15]. Instances of these constraints arise frequently in practice such as network tomography and interactive content search.

Network Tomography

A frequent situation that arises in sensor network tomography can be briefly explained as follows [4]. Suppose we are to remotely maintain a sensor network, whose nodes are severely restricted. First, being connected in a graph G, they can communicate with their neighbours only. Second, although they can receive broadcast messages from a base station, communication from any node back to the base must be kept to an absolute minimum due to power constraints. As nodes fail regularly, it is necessary to periodically search for faults. For simplicity, we assume that at most one node is faulty. To minimize the communication back to base, we should use the least number of broadcasts of the form "To nodes in Q: Is one of you faulty?". If Q is a connected subgraph of G, the question can be answered by a simple local message passing protocol, where a designated root in Q reports positively to the base only if it receives messages from all neighbours in Q, etc. Given that this does not happen, the root can poll the branch in question, whereby the faulty node can be identified recursively and passed back to the root, which reports it back to base. Finally, if the base does not get any reply after a reasonable time, it concludes that the chosen root must be faulty.

We address the constraints of this type under the general and realistic notion of *graph constraints* where we assume that the set $\mathcal{N}$ is furnished with an undirected graph structure G, and $Q \subset \mathcal{N}$ is an admissible query if and only if the subgraph $G|_Q$ induced by Q (containing all edges of G between vertices in Q) is connected. We then provide an efficient constant factor approximation to the "graph-constrained" twenty questions problem, which finds the target in $\leq 4H[P] + 2$ queries on average.

Interactive Content Search

In content search applications with humans involved, it is of obvious value to keep the number of queries as small as possible, as human interventions are disproportionally costly. Examples include visual recognition [2] and pattern classification [11], where questions are restricted to simple visual attributes. Whenever people are in the loop, queries are limited by their ability to meaningfully disambiguate in the presence of many attributes. As these examples show, it is usually not practical to allow for *any* query $Q \subset \mathcal{N}$. In these scenarios, it is more natural to define the constraints in terms of a cost rather than through a graph.

More formally, we are given a *non-decreasing cost function* $C : \{1, 2, \cdots, n\} \to \mathbb{R}$ where the cost of making an oracle query on set S is $C(|S|)$. Notice that the cost only depends on the size of the set and not the elements it contains. When a human plays the role of the membership oracle, it is more difficult to answer a query with a bigger set than one with a smaller set [2]. This is why in many interactive search strategies such a cost function arises naturally. In this case, we provide an approximation algorithm that can find the target within $O(\log(\log n))$ gap from the cost of the optimum algorithm.

To generalize our analysis beyond non-decreasing cost functions, we consider the case where constraints are defined in terms of a *submodular function* $Cost : 2^{\mathcal{N}} \to \mathbb{R}$. In this setting, for all subsets $S_1 \subseteq S_2 \subseteq \mathcal{N}$ and an item $i \in \mathcal{N}$ we have $Cost(S_1 \cup \{i\}) - Cost(S_1) \geq Cost(S_2 \cup \{i\}) - Cost(S_2)$. In words, adding a new item i to a larger set should produce an incremental cost no more than adding i to a smaller set. Under submudular constraints, we provide an approximation algorithm that can find the target within $O(\log(n))$ gap from the cost of the optimum algorithm. Further applications of submodular functions in active learning can be found in [12].

2 Related Work

The problem studied here can be seen as a special case of the binary identification problem (BIP) [10]. Suppose that we are given a set of objects $\mathcal{N}$ and a set of tests $\{t_1, \dots . t_k\}$ where one of the objects is marked. Each test determines whether the marked object is in the test set or not. The goal is to define a strategy that minimizes the number of tests to find the marked object. It is known that both the average case minimization and worse case minimization are NP-complete [15]. Moreover, it is even NP-Hard to have an $o(\log n)$-approximation algorithm for the average case [5]. In both cases, there exist heuristic algorithms that admit $O(\log n)$-approximation [20, 6]. The closest work to ours is [14] where the authors study the same problem in a setting that a cost is associated with each test and the goal is to minimize the average total cost. They propose an $O(\log n)$-approximation algorithm. Unfortunately, there is no straight way to follow their approach and obtain similar performance guarantees for the submodular cost functions (for the other two problems that we consider in this paper, our approximation factors are better). Note that in this case, there are 2^n number of tests and a naive reduction admits an exponential running time. To overcome this barrier we propose a novel algorithm with a similar approximation guarantee.

Adding some structure to the set of tests leads to interesting special cases. For instance, if we let the set of tests be the power set of objects, then the optimal average case strategy is attained by Huffman coding [7]. Another basic variant of BIP is finding a marked element in a totally ordered set, a problem that is very well studied in the literature [19]. This can be generalized to searching in structures where the input has a partial order between its elements instead of a total order [1, 3]. It is known that searching in posets for the worst case minimization is NP-hard [3]. However, in [20], the authors showed a greedy algorithm with $O(\log n)$ approximation which was further improved to a constant factor approximation by [21]. It has been recently shown in [17] that the average case minimization is also NP-hard for the class of trees with diameter at most 4.

Constrained BIP is strongly related to *active learning* [8, 22, 13, 9, 18] in which a *hypothesis space* $\mathcal{H}$ is defined as a set of binary valued functions defined over a finite set $\mathcal{Q}$, called the *query space*. Each hypothesis $h \in \mathcal{H}$ generates a label from $\{-1, +1\}$ for every query $q \in \mathcal{Q}$. A target hypothesis h^* is sampled from $\mathcal{H}$ according to some prior P; asking a query q amounts to revealing the value of $h^*(q)$, thereby restricting the possible candidate hypotheses. The goal is to determine h^* by asking as few queries as possible. In our setting, the hypothesis space $\mathcal{H}$ is the set $\mathcal{N}$, and the query space $\mathcal{Q}$ is the set of all admissible subsets. The target hypothesis sampled from P is the target t_*. A well-known algorithm for determining the true hypothesis in the active-learning setting is the *generalized binary search* (GBS) or *splitting* algorithm [8, 22, 9]. Define the *version space* $V \subseteq \mathcal{H}$ to be the set of possible hypotheses that are consistent with the query answers observed so far. At each step, GBS selects the query $q \in \mathcal{Q}$ that minimizes $|\sum_{h\in V} P(h)h(q)|$. Recently, Golovin and Krause [12] showed that GBS makes at most $OPT \cdot \big(H_{\max}(\mu) + 1\big)$ queries in expectation to identify hypothesis $h^* \in \mathcal{N}$, where OPT is the minimum expected number of queries made by any adaptive policy and $H_{\max}(P) = \max_{x\in\text{supp}(P)} \log \frac{1}{P(x)}$. In our setting, the version space V comprises all possible objects in $z \in \mathcal{N}$ that are consistent with answers given so far. Under graph, cost, and submodular constraints, we replace $H_{\max}$ (which in practice can be

Algorithm 1 The role of size

Input: Connected constraint graph $G = (\mathcal{N}, \mathcal{E})$, root node $v_R \in \mathcal{N}$.
Grow a connected subtree $T \subset G$, starting from v_R, by a breadth-first search (BFS), until T spans $\lceil n/2 \rceil$ vertices.
Query the vertex set of T (size $\lceil n/2 \rceil$).
if $\mathrm{I}_{\{t_* \in T\}} = 1$ **then**
  Recurse on T, root v_R.
else
  Collape all nodes in T into a single node v_T, which is connected to all $v \in \mathcal{N}$ adjacent to nodes in T in the original G. Create G' by Connecting all neighbours of v_T to one another and removing v_T. Recurse on G'.
end if

quite large) by a constant, $O(\log(\log(n)))$, and $O(\log(n))$, respectively.

The use of interactive methods for searching in a dataset has a long history in literature. Relevance feedback [24] is a method for interactive image retrieval, in which users mark the relevance of image search results, which then used to create a refined search query. We use the membership oracle to model the role of a user for identifying a target in a database. In practice, the cost of a query depends on the characteristics of the query, e.g., its size [11, 2]. However, due to the lack of analytical results, in many such applications only heuristic methods were proposed. To close this gap, we introduce general constraints, in terms of graph, cost and submodular functions, on the set of queries and establish analytical guarantees associated with them.

3 Graph Constraints

Let us assume that the set $\mathcal{N}$ is endowed with undirected graph structure, $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{E}$ contains the edges. For any subset $A \subset \mathcal{N}$, $G - A$ denotes the graph obtained from G by removing all vertices A and all edges adjacent to A, while $G|_A$ is the graph induced by A (vertex set A, edge set those $e \in \mathcal{E}$ between vertices in A).

Given G, a query $Q \subset \mathcal{N}$ is *admissible* if and only if the graph $G|_Q$ is connected. Our detection algorithm can submit any admissible query Q to a membership oracle, which initially sampled the target t_* at random from P: it will answer by one bit of information, $\mathrm{I}_{\{t_* \in Q\}} = 1$ if $t_* \in Q$, 0 otherwise. Our goal is to detect t_* with as few queries to the oracle as possible. Here, we restrict ourselves to deterministic policies which terminate only once the version space for t_* consistent with all previous queries $Q_1, \dots, Q_k$ (formally, $(\cap_{k | t_* \in Q_k} Q_k) \cap (\cap_{k | t_* \notin Q_k} (\mathcal{N} \setminus Q_k))$) reaches size one. Throughout this section, we assume that the algorithm knows the distribution P the target is drawn from.

We begin with Algorithm 1, a simple divide-and-conquer scheme which detects the target with no more than $\lceil \log n \rceil$ queries, if seeded with an arbitrary $v_R \in \mathcal{N}$ as root node. Here and elsewhere, we use binary logarithms (base 2). Essentially, this algorithm mirrors the principle of binary search by way of intermediate sets produced during a breadth-first-search (BFS). Given adequate backpointer structures, its running cost is that of a single BFS.

▶ **Lemma 1.** *Presented with a connected constraint graph $G = (\mathcal{N}, \mathcal{E})$, where $|\mathcal{N}| = n$, Algorithm 1 detects the target t_* with no more than $\lceil \log n \rceil$ queries to the membership oracle.*

While Algorithm 1 is simple and efficient, it has the obvious drawback of not exploiting knowledge about the distribution P at all. However, it is useful as subroutine for our main algorithm developed next.

Suppose w.l.o.g. that $P = (p_1, \ldots, p_n)$ is such that $p_1 \geq p_2 \geq \cdots \geq p_n$. Define the nested subsets $A_1 \subset A_2 \subset \cdots \subset \mathcal{N}$ by

$$A_i = \{1, \ldots, j_i\}, \quad j_i = \min \left\{ j \;\middle|\; \sum\nolimits_{k>j} p_k \leq 2^{-i} \right\}.$$

Put differently, A_i is the smallest subset of $\mathcal{N}$ so that $\sum_{k \in A_i} p_k \geq 1 - 2^{-i}$. Also, define $a_i = \log(1/p_{j_i})$, so that $p_k \geq 2^{-a_i}$ for all $k \in A_i$, moreover $a_0 = 0$. The intuition behind this choice is that A_1 contains about half of the probability mass, $A_2 \setminus A_1$ half of the remaining mass, and so on.

Our algorithm processes the A_i in order, $i = 1, 2, \ldots$. For A_i, it calls Algorithm 1, which will detect t_* if it is in A_i, otherwise we move to the next set. However, we cannot simply pass the induced subgraph $G|_{A_i}$ to Algorithm 1, as it may well not be connected. In this case, we generate $G_i = (A_i, V_i)$ passed to the algorithm as follows. We initialize $G_i \leftarrow G|_{A_i}$ and cluster the nodes into connected components. We then join each pair of disjoint components by a shortest path π, a central part of which necessarily features nodes $V(\pi)$ with $V(\pi) \cap A_i = \emptyset$. We collapse this part of π into a new *virtual* edge between nodes in A_i, which is labelled by the vertices $V(\pi)$ and added to V_i. This process stops when G_i is connected. Finally, when running Algorithm 1 on G_i, we need to translate queries back to connected subgraphs of the original G. For a query $Q \subset A_i$ in question, this is done by adding nodes with which virtual edges of $G_i|_Q$ are labelled. Our construction of G_i from A_i and labeling of virtual edges ensures that any query processed in this way corresponds to a connected subgraph of G, therefore is admissible.

Note that due to the presence of virtual edges, Algorithm 1 may receive positive answers from the oracle, even though $t_* \notin A_i$. After all, t_* might be among the vertices on virtual edges. However, in this case, Algorithm 1 simply descends to a single vertex $v_i \in A_i$, so that one more query $\{v_i\}$ settles the question "$t_* \in A_i$". Our main result is as follows.

▶ **Theorem 2.** *Given a connected constraint graph G with n vertices and any distribution P over $\{1, \ldots, n\}$, our algorithm finds a target t_* sampled at random from P with no more than $2 + 4\mathrm{H}[P]$ admissible queries, on average over draws of t_*.*

Proof. We prepare our proof with a lemma whose proof can be found in the full version.

▶ **Lemma 3.** *For the numbers a_k defined above, we have that $\sum_{k=1}^{\infty} a_k / 2^{k+1} \leq \mathrm{H}[P]$.*

To establish Theorem 2, recall that we visit sets A_i in order, $i = 1, 2, \ldots$, calling Algorithm 1 on $G_i = (A_i, V_i)$ constructed as detailed above. Since $p_j \geq 2^{-a_i}$ for all $j \in A_i$, the size of A_i is bounded by 2^{a_i}, and Algorithm 1 returns after $\leq a_i$ queries. As noted above, due to "virtual edge" complications, we may have to query one more single node, yet after $\leq a_i + 1$ questions we know whether $t_* \in A_i$ or not, and in the former case will have detected t_*. Now, the probability of not finding t_* in A_i or earlier is bounded by $\sum_{j | p_j < 2^{-a_i}} p_j \leq 2^{-i}$. This means that the expected number of queries in in our algorithm is at most $\sum_{i \geq 1} (a_i + 1)/2^{i-1} \leq 2 + 4\mathrm{H}[P]$, where this inequality is due to Lemma 3. ◀

4 Cost Function Constraints

In this section we analyze another variant of the binary identification problem where the constraints are defined in terms of a cost function rather than a graph. More formally, we are given a non-decreasing cost function $C : \{1, 2, \cdots, n\} \to \mathrm{R}$ where the cost of making an

Algorithm 2 Binary Identification Algorithm with Cost Function Constraints

Input: n objects with a probability distribution on them, and a cost function C : $\{1, 2, \cdots, n\} \to \mathrm{R}$, fixed constant $\epsilon > 0$.

Create clusters $S_1, S_2, \dots S_l$ for $l = \log(n^2/\epsilon)$.

(Phase one) Use Procedure 3 to determine which cluster contains the target t_*.

(Phase two) If cluster S_i contains the target, find it by using the dynamic program 2.

(Phase three) If the target is not found in any of the above clusters, query each of the non-clustered objects.

oracle query on set S is $C(|S|)$[1]. Notice that the cost only depends on the size of the set not the elements it contains. When a human plays the role of the membership oracle, it is more difficult to answer a query with a bigger set than one with a smaller set. This is why in many interactive search strategies such a cost function arises naturally. To avoid confusion in using term "cost" for queries and algorithms, we formally define the notion of cost for sets and algorithms as follows.

▶ **Definition 4.** We refer to the cost of making a query on a set S by the "cost of set S". On the other hand, we use the term "expected search cost" of an algorithm to represent the expected value of total cost of queries the algorithm makes. Formally, An algorithm A consists of a family of possible queries $\mathcal{F}_A$. Algorithm A asks each query $S \in \mathcal{F}_A$ with probability $\Pr(A, S)$ which is a function of both the algorithm A, and the probability distribution of objects, P. We define the expected search cost of algorithm A to be $\sum_{S \in \mathcal{F}_A} \Pr(A, S) C(|S|)$.

By the above definition, it makes sense to talk about the expected search cost of Algorithm 2 or any other algorithm such as the optimum algorithm. We can also refer to the expected cost associated with a part of Algorithm 2 (it has three phases) and we can similarly define its expected cost in each phase. Note that the expected search cost of an algorithm is exactly the expected value of its total cost according to distribution P.

▶ **Definition 5.** Let $\epsilon > 0$ to be a small and fixed constant. We place objects in $l = \log(n^2/\epsilon)$ clusters based on their probabilities as follows. Let S_i be the cluster of objects with probability in range $(1/2^i, 1/2^{i-1}]$ for $1 \le i \le l$. For any $1 \le i \le l$, we define $\Pr(S_i)$ to be the sum of probabilities of objects in cluster S_i, and we define $\Pr[i, j] \doteq \sum_{x=i}^{j} \Pr(S_x)$ for $1 \le i \le j \le l$. We let $\Pr[i, j] = 0$ for $i > j$.

The choice of ϵ in the above definition is for ensuring that non-clustered objects have negligible probabilities, namely, at most ϵ/n^2.

Our algorithm for finding the target is shown in Algorithm 2. In phase one, we run a recursive algorithm that uses procedure *ClusterFinder* (shown in Algorithm 3) as a subroutine. The procedure *ClusterFinder* gets two numbers i and j, and finds the cluster containing the target only if the target is in one of the clusters $\{S_i, S_{i+1}, \cdots, S_j\}$. More specifically, Algorithm 3 finds a number k and a collection of clusters Q to query as follows:

$$Q = \bigcup_{i \le m \le k} S_m, \quad k = \max \left\{ k' \;\middle|\; \Pr[i, k'] < \frac{\Pr[i, j]}{2} \right\} \tag{1}$$

[1] Note that there are two ways to determine whether or not the target lies in some set S. We can query set S or we can query the complement set $\mathcal{N} \setminus \mathcal{S}$. Here, we assume that the cost is always a function of the size of the queried set and it is the job of the algorithm to determine which set needs to be queried.

Algorithm 3 ClusterFinder(i,j)

Input: Clusters $S_i, S_{i+1}, \cdots, S_j$

1: Find the number k and the set Q according to Equation 1. Query set Q.
2: If $t_* \in Q$ call $ClusterFinder(i, k)$. Otherwise, query S_{k+1}.
3: If $t_* \in S_{k+1}$ return S_{k+1}. Otherwise, call $ClusterFinder(k+2, j)$.

If the target is in Q, the algorithm calls procedure $ClusterFinder(i, k)$. Otherwise, it queries set S_{k+1}, and if the target is not there either, it calls $ClusterFinder(k+2, l)$. Note that in Eq. 1, we might have $\Pr(S_i) = \Pr[i, i] \geq \Pr[i, j]/2$ which causes k to be $i-1$. In this case, the set Q will be $\emptyset$.

▶ **Definition 6.** The procedure calls of Algorithm 3 can be represented by a binary tree T as follows. The root node is the procedure $ClusterFinder(1, l)$. The left child of the root is $ClusterFinder(1, k)$, and its right child is $ClusterFinder(k+2, l)$. In the same fashion, we can define the rest of the tree nodes based on the recursive calls.

In phase two of Algorithm 2, we are given a cluster S_i that contains the target, and we want to find the target object. Let x be the number of objects in S_i. Our algorithm assumes that the probabilities of all objects in S_i are identical. Since they are in the same cluster, their probabilities are close to each other (we prove later that this assumption does not incur much cost for us). By this assumption we have symmetry among objects in cluster S_i. Therefore, the only relevant characterization of a subset of S_i is its size. We define a dynamic programming array $A[j]$, for $1 \leq j \leq x$, which is the expected search cost of optimal strategy to find the target given that the target is in a subset of S_i with size j under the assumption that these objects have the same probability, and therefore are identical. We can fill up the entries of array A as follows. It is clear that $A[1]$ is zero. We can update each $A[j]$ using the lower entries $A[1], A[2], \cdots, A[j-1]$ as follows:

$$A[j] = \min_{1 \leq j' < j} \{C(j') + A[j'](j'/j) + A[j-j']((j-j')/j)\}. \tag{2}$$

The optimal strategy chooses some $1 \leq j' < j$, and a subset of size j' among the remaining objects (it does not matter which subset it chooses, the only important factor is the size of the subset because of the identical probabilities assumption). Making a query on the selected subset of size j' has cost $C(j')$. With probability j'/j, the target is in the selected set, and we have to pay $A[j']$ in this case. With probability $(j-j')/j$, the target is not in the selected set, and we have to pay $A[j-j']$ in this case. Since we assume that S_i contains x elements, the optimal cost for finding the target in cluster S_i is therefore $A[x]$.

Finally, phase three of Algorithm 2 makes sure that if we have not found the target in the previous phases, we query each nonclustered object until we find the target.

▶ **Theorem 7.** *Let C_{Greedy} and C_{OPT} denote the expected search costs of Algorithm 2 and the optimum algorithm, respectively. Then, we have $C_{Greedy} = O(\log(\log(n))C_{OPT})$.*

Before we proceed to the proof of this theorem, let us consider the following definitions for *general* cost functions. Note that in general, the cost of a query is a function of the query set and not necessarily its size.

▶ **Definition 8.** Let $I(X)$ denote an instance of the search problem on a subset $X \subset \mathcal{N}$ where the probabilities of objects in X are normalized to make sure their sum is one, and we know that the target is in X. We denote by $Cost(X')$ the cost of making a query on

$X' \subset X$. Let also $Opt(I(X))$ represent the expected search cost of the optimum algorithm for instance $I(X)$. We define

$$Cost^i_X = \min\left\{Cost(X')|X' \subseteq X, \Pr(X') \geq 1 - 1/2^i\right\}.$$

Let $S(X,i)$ denote the subset for which $Cost(S(X,i)) = Cost^i_X$ and $\Pr(S(X,i)) \geq 1 - 1/2^i$.

Note that the particular cost function we consider in this section is $Cost(X') = C(|X'|)$. The following lemma will provide us with a general lower bound on the expected search cost of the optimum solution.

▶ **Lemma 9.** $Opt(I(X)) \geq \sum_{i=1}^{\infty} Cost^i_X/2^{i+1}$.

Lemma 9 help us bound the expected search cost of the optimum algorithm from below.

Proof of Theorem 7. We first show that the expected search cost caused by nonclustered objects is negligible.

▶ **Lemma 10.** *The expected search cost incurred by the nonclustered objects in phase three of Algorithm 2 is at most $\epsilon \cdot C_{OPT}$.*

Proof. The probability of each nonclustered objects is at most $1/2^l = \epsilon/n^2$. There are at most n non-clustered objects, so the probability of the target is one of these non-clustered objects is at most ϵ/n. With probability at most ϵ/n, we make a query for each non-clustered object. Hence, its expected search cost is at most $\epsilon n \times nC(1) = \epsilon C(1)$. On the other hand, note that $C(1)$ is a lower bound for the optimum solution because no matter where the target is we always have to make at least a query of size at least one to find the target. ◀

Lemma 10 entails that the expected search cost incurred by nonclustered objects is much less than the optimal expected search cost. Thus for simplicity we can assume that there exists no nonclustered object. In order to bound the expected search cost of the first phase of Algorithm 2 we need a few intermediate results. Let us start with the following definition.

▶ **Definition 11.** A set of nodes S of a tree T' is sparse, if and only if there do not exist three nodes v_1, v_2 and v_3 in S such that v_1 is one of the ancestors of v_2 and v_3, and neither of v_2 and v_3 is an ancestor of one another.

Hence, in a sparse set S there cannot be two nodes such that they don't have an ancestor/-descendant relationship and simultaneously have a common ancestor in S. The following lemma bounds the number of possible sparse partitions of any tree

▶ **Lemma 12.** *The nodes of any tree T' can be partitioned into $\lceil \log(|T'|) \rceil$ sparse sets where $|T'|$ is the number of nodes in T'.*

To link the expected search cost of the first phase of Algorithm 2 to nodes of the tree T, we need the following definition.

▶ **Definition 13.** Let a node $v \in T$ represent procedure $ClusterFinder(i,j)$ for some $1 \leq i \leq j \leq l$. With probability $\Pr[i,j]$, we go to this node, and we make a query on the set Q as defined in Eq. 1. If the target is not in Q (i.e., it is in one of the clusters $S_{k+1}, S_{k+2}, \cdots, S_j$), we make a second query on the set S_{k+1}. Hence, the expected search cost of our algorithm at node v is $\Pr[i,j]C(|Q|) + \Pr[k+1,j]C(|S_{k+1}|)$. We define this quantity as the price of node v.

It is clear that the expected search cost of our algorithm in phase 1 is the sum of the prices of all nodes in tree T. In Lemma 14, we bound the sum of prices of nodes of a sparse set which provide us with the key result to bound the expected search cost of the first phase. The proof of Lemma 14 heavily relies on the observation we made in Lemma 9.

▶ **Lemma 14.** *The sum of prices of nodes of a sparse set is $O(C_{OPT})$.*

Due to the above lemma, we know that the total prices of the nodes of every sparse set is $O(C_{OPT})$. By recalling Lemma 12 we also know that we can partition the nodes of T into $O(\log(\log(n)))$ sparse sets. Hence, Lemma 14 together with Lemma 12 readily bounds the expected search cost of the first phase which is the sum of prices of all nodes in tree T.

▶ **Lemma 15.** *The expected search cost for identifying which cluster contains the target t_* (first phase of Algorithm 2) is $O(\log(\log(n)) \cdot C_{OPT})$.*

We are ready to analyze the second phase of algorithm 2. In this part, we know the cluster, say S_i, that contains the target t_* and we just need to find it. Let us define I_i to be this instance: we are given the information that the target is in S_i, and we have to find it. Following Definition 8, we denote the optimum expected search cost to identify the target in I_i by $OPT(I_i)$. The following lemma bounds the expected search cost of Algorithm 2 (phase two) in terms of $OPT(I_i)$.

▶ **Lemma 16.** *Assume that the target t_* is in cluster S_i. Then, the expected search cost of phase two of Algorithm 2 to find the target is at most $2\Pr(S_i)OPT(I_i)$.*

Lemma 16 helps us relate the expected search cost of phase two to C_{OPT}.

▶ **Lemma 17.** *The expected search cost of phase two of Algorithm 2 is at most $2C_{OPT}$.*

By combining Lemmas 10, 15, and 17 we can conclude that the expected search cost of Algorithm 2 is $O(\log(\log(n)) \cdot C_{OPT})$. ◀

5 Submodular Constraints

In this section we analyze another variant of the binary identification problem where the constraints are defined in terms of a submodular function. More specifically, we are given a set of objects $\mathcal{N} = \{1, 2, \cdots, n\}$ and a non-decreasing submodular function $C_{sub} : \mathcal{F}_n \to \mathbb{R}$ where $\mathcal{F}_n = 2^{\mathcal{N}}$ is the family of all subsets of $\mathcal{N}$. We denote the cost of making a query on the set $S \subset \mathcal{N}$ by $C_{sub}(S)$ where we assume that the cost function is submodular:

$$\forall S_1 \subseteq S_2 \subseteq U,\ i \in U : C_{sub}(S_1 \cup \{i\}) - C_{sub}(S_1) \geq C_{sub}(S_2 \cup \{i\}) - C_{sub}(S_2).$$

Intuitively, this property insures that adding an object i to a set S_1 increases its cost at least as much as adding i to a superset S_2. Another equivalent definition that we later use reads as follows: $\forall S_1, S_2 \subseteq \mathcal{N}, C_{sub}(S_1) + C_{sub}(S_2) \geq C_{sub}(S_1 \cup S_2) + C_{sub}(S_1 \cap S_2)$. In view of Definition 8 the cost function of a subset $S \in \mathcal{N}$ is $Cost(S) = C_{sub}(S)$.

Before elaborating on our new algorithm, we should note that it is possible to mimic the first phase of Algorithm 2 and obtain an approximation factor of $O(\log(\log(n)))$. However, it is no longer possible to run the second phase with the same approximation factor for submodular functions. Recall that in the second phase, we relied on a dynamic program that only depends on the size of the queried sets. Due to the fact that in our new setup, the cost of a query depends on the set (an not only on its size), we need to reformulate the dynamic program: it matters which subset is chosen. To do so, we ought to construct an

Algorithm 4 Bicriteria Approx. Alg.

Input: $\epsilon > 0$.
1: Find all α_k's in (3) and their corresponding sets S_{α_k}.
2: Among all S_{α_k}, keep only those that have $\Pr(S_{\alpha_k}) \geq 1/6$.
3: Among the remaining sets, choose the one with minimum $C_{sub}(S_{\alpha_k})$. Call this set S'.
4: Keep removing objects from S' while its probability remains at least $1/6$.

Algorithm 5 BIP with Submod. Cnst.

Input: Set of objects $\mathcal{N}$.
1: Find set $S' \subset \mathcal{N}$ (and $\bar{S}' = \mathcal{N} \setminus S'$) by using Algorithm 4. Query S'.
2: **if** set S' contains the target t_* **then**
3: Query one of the objects $i \in S'$. Either i is the target, or otherwise recurs on $S' \setminus \{i\}$.
4: **else**
5: Recurs on $\bar{S}'$.
6: **end if**

exponential size array (one for each subset) and as a result, the algorithm will no longer run in polynomial time. To avoid this drawback, we can devise a $\log(n)$-approximation algorithm for the second phase of Algorithm 2 which can incorporate submodular constraints. Instead, in this section we present a much simpler algorithm with the same approximation guarantee. To do so, we first need an intermediate step.

Submodular Functions Under Linear Constraints

The problem of "minimizing submodular functions under linear constraints" is to find a subset $S^* \in \mathcal{N}$ such that $\Pr(S^*) \geq 1/2$ and its cost, namely $C_{sub}(S^*)$, is minimum. Here, we present a bicriteria approximation algorithm that finds a set S' with $C_{sub}(S') \leq 3C_{sub}(S^*)$ and $\Pr(S') \geq 1/6$. To do so, we first need to define another submodular function $f^\alpha : 2^{\mathcal{N}} \to \mathbb{R}$ for which $f^\alpha(S) = C_{sub}(S) + \alpha \cdot \Pr(\bar{S})$, where $\bar{S} = \mathcal{N} \setminus S$, and $\alpha > 0$ is a real number. The reason that f^α is a submodular function simply follows from the fact that we only added a liner term to the submodular function C_{sub} (check the equivalent definition we provided earlier). It is a folklore result that submodular function minimization problem has strongly polynomial time exact algorithms due to [23] and [16]. Hence, one can find a subset $S_\alpha \subseteq \mathcal{N}$ in polynomial time that admits the minimum value of f^α. Let $c_{\min} = \min_{i \in \mathcal{N}} C_{sub}(\{i\})$. For a fixed $\epsilon > 0$ we also define

$$\alpha_k = 3c_{\min}(1+\epsilon)^k, \quad \text{for all } k = 0, 1, \ldots, \lceil \log_{1+\epsilon}(C_{sub}(\mathcal{N}))/c_{\min} \rceil. \tag{3}$$

Note that there exists a $k = \kappa$ for which $3C_{sub}(S^*) \leq \alpha_\kappa \leq 4C_{sub}(S^*)$.

▶ **Lemma 18.** *Let S_{α_κ} denote the set that admits the minimum of the function f^α for $\alpha = \alpha_\kappa$. Then $C_{sub}(S_{\alpha_\kappa}) \leq 3C_{sub}(S^*)$ and $\Pr(S_{\alpha_\kappa}) \geq 1/6$.*

Since the value of $C_{sub}(S^*)$ is not known a priori, we cannot apply Lemma 18 without modification. To this end, we propose Algorithm 4. It first calculates all values of α_k. For each α_k it finds the corresponding set S_{α_k} that minimizes f^{α_k}. It then finds among those S_{α_k} with $\Pr(S_{\alpha_k}) \geq 1/6$, the one that has minimum cost $C_{min}(S_{\alpha_k})$. Once such a set S' is found, it starts removing objects from S' while keeping $\Pr(S') \geq 1/6$.

▶ **Lemma 19.** *Algorithm 4 outputs a set $S' \subset \mathcal{N}$ with $C_{sub}(S') \leq 3C_{sub}(S^*)$ and $\Pr(S') \geq 1/6$. Moreover, for any $i \in S'$, we have $\Pr(S' \setminus \{i\}) < 1/6$.*

Approximation Algorithm with Submodular Constraints

Algorithm 5 starts by finding a set $S' \subset \mathcal{N}$. To this end, it calls on Algorithm 4. According to Lemma 19, for the set S' we have $\Pr(S') \geq 1/6$ and $C_{sub}(S') \leq 3C_{sub}(S^*)$. Remember S^*

is the set with minimum cost and the probability at least a half. Ideally, we would like to query the set S^*. Unfortunately, it is not easy to find such a set. Instead, Algorithm 5 queries the set S' that approximate our ideal candidate. If the target t_* is in S', then Algorithm 5 chooses an arbitrary object $i \in S'$ and see whether i is the target or not. In case it is not the target, the whole procedure repeats now from the set $S' \setminus \{i\}$, namely, Algorithm 4 will be called for the set $S' \setminus \{i\}$. If the target is not in S' from the beginning, the algorithm recurs on set $\bar{S}' = \mathcal{N} \setminus S'$. Note that by making a singleton query before recursion, Algorithm 5 makes sure that the probability of the remaining set will shrink by a factor of $1/6$ (Lemma 19) if the target is in set $S' \setminus \{i\}$. Otherwise, the probability shrinks by a factor of $5/6$ because we have that $\Pr(\bar{S}') = 1 - \Pr(S') \leq 1 - 1/6$.

▶ **Theorem 20.** *Let C_{Greedy} and C_{OPT} denote the expected search costs of Algorithm 5 and the optimum algorithm, respectively. Then, we have $C_{Greedy} = O(\log(n) \cdot C_{OPT})$.*

Proof. The proof of this theorem is similar, *mutatis mutandis*, to the proof of phase one of Theorem 7. Let us first modify Definition 8 as follows. Let $Cost^i_X = \min_{X' \subseteq X | Pr(X') \geq 1-(5/6)^i} Cost(X')$. Then, we can show

▶ **Lemma 21.** $Opt(I(X)) \leq \sum_{i=1}^{\infty} \frac{Cost^i_X}{(5/6)^{i+1}/5}$.

The search mechanism portrayed in Algorithm 5 defines a binary tree T, similar to the one we saw in the previous section. In brief, the tree starts from the root $\mathcal{N}$. In each internal node S with probability at least $1/6$ we go to the left child (representing set $S' \setminus \{i\}$) and with the remaining probability we go to the right child (representing $S \setminus S'$). Note that there are at most $2n$ nodes in this tree where the probability of each child is at most $5/6$ of the probability of its father (instead of $1/2$ in the previous section). Because the tree T has at most $2n$ nodes, instead of $O(\log\log(n))$ sparse sets, we need $\log(2n)$ sparse sets to cover the expected cost of all nodes. As a result one can provide a similar proof showing that the expected cost of Algorithm 5 with submodular constraints is at most $O(\log(n))$ times the cost of the optimum solution.

◀

6 Conclusion

In this work we considered the problem of binary identification problem (BIP) under the general notion of graph, cost and submodular constraints. We also provided novel polynomial time approximation algorithms with provable analytical guarantees. Even though we believe that the three variants of BIP we considered are all NP-hard, we did not provide any rigorous proof. This is an interesting future direction we would like to pursue.

References

1 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees, 1999.

2 Steve Branson, Catherine Wah, Florian Schroff, Boris Babenko, Peter Welinder, Pietro Perona, and Serge Belongie. Visual recognition with humans in the loop. In *ECCV (4)*, pages 438–451, 2010.

3 R. Carmo, J. Donadelli, Y. Kohayakawa, and E. Laber. Searching in random partially ordered sets. *Theor. Comput. Sci.*, 321:41–57, 2004.

4 R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical Science*, 19(3):499–517, 2004.

5 Venkatesan T. Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh K. Mohania. Decision trees for entity identification: Approximation algorithms and hardness results. *ACM Transactions on Algorithms*, 7(2):15, 2011.

6 Venkatesan T. Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, and Yogish Sabharwal. Approximating decision trees with multiway branches. In *ICALP (1)*, pages 210–221, 2009.

7 Thomas M. Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

8 S. Dasgupta. Analysis of a greedy active learning strategy. *NIPS*, 2005.

9 M. R. Garey and R. L. Graham. Performance bounds on the splitting algorithm for binary testing. *Acta Informatica*, 3:347–355, 1974.

10 M.R. Garey. Optimal binary identification procedures. In *SIAM J. Appl. Math*, 1972.

11 Donald Geman and Bruno Jedynak. Shape recognition and twenty questions. Technical report, in Proc. Reconnaissance des Formes et Intelligence Artificielle (RFIA, 1993.

12 Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization. *Journal of Artificial Intelligence Research (JAIR)*, 42:427–486, 2011.

13 Andrew Guillory and Jeff A. Bilmes. Average-case active learning with costs. In *ALT*, pages 141–155, 2009.

14 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 690–701, Berlin, Heidelberg, 2010. Springer-Verlag.

15 L. Hyafil and R. Rivest. Constructing optimal binary decision trees is np-complete. In *Information Processing Letters*, pages 15–17, 1976.

16 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

17 Tobias Jacobs, Ferdinando Cicalese, Eduardo Laber, and Marco Molinaro. On the complexity of searching in trees: average-case minimization. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 527–539, 2010.

18 Amin Karbasi, Stratis Ioannidis, and Laurent Massoulie. Comparison-based learning with rank nets. In *29th International Conference on Machine Learning (ICML)*, 2012.

19 D. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley Longman Publishing Co., Inc, 1998.

20 S. Rao Kosaraju, Teresa M. Przytycka, and Ryan S. Borgstrom. On an optimal split tree problem. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, WADS '99, pages 157–168. Springer-Verlag, 1999.

21 Eduardo Laber and Marco Molinaro. An approximation algorithm for binary searching in trees. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 459–471, Berlin, Heidelberg, 2008. Springer-Verlag.

22 R.D. Nowak. The geometry of generalized binary search. *Transactions on Information Theory*, 5, 2012.

23 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000.

24 Xiang S. Zhou and Thomas S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, 2003.

A Proofs

In this section, we provide all the proofs of the lemmas we used to establish our main results.

Proof of Lemma 1

The bound on the number of queries follows directly from the fact that each query splits the remaining version space in half. During the run of the algorithm, the root node v_R may be a “meta-node”, representing a connected subgraph of what we started from, which we need to expand every time we submit a query containing v_R. From a different viewpoint, running Algorithm 1 consists of a single BFS scheduled in the right way. At a point where the intermediate BFS tree T of size $\lceil n/2 \rceil$ results in $\mathrm{I}_{\{t_* \notin T\}}$, we simply continue the BFS until size $\lceil (3/4)n \rceil$, and so on.

Proof of Lemma 3

For any $i \geq 1$, define $T_i = \{j \mid 2^{-a_{i+1}} < p_j \leq 2^{-a_i}\}$, moreover $Q_i = \sum_{j \in T_i} p_j$. By definition, $a_i \leq \log(1/p_j)$ for any $j \in T_i$, so that $\mathrm{H}[P] \geq \sum_{i \geq 1} Q_i a_i$. Moreover,

$$\sum_{i \geq 1} Q_i a_i = \sum_{i \geq 1} Q_i \sum_{k=1}^{i} (a_k - a_{k-1}) = \sum_{k \geq 1} (a_k - a_{k-1}) \sum_{i \geq k} Q_i.$$

By definition of the Q_i, we have that

$$\mathrm{H}[P] \geq \sum_{k \geq 1} (a_k - a_{k-1}) \sum_{j \mid p_j \leq 2^{-a_k}} p_j \geq \sum_{k \geq 1} (a_k - a_{k-1}) 2^{-k} = \sum_{k \geq 1} a_k / 2^{k+1}.$$

Here, the second inequality is due to the definition of a_k and associated sets A_k.

Proof of Lemma 9

In the optimum solution to this instance, $I(X)$, let Q_1 be the first query set, and if the target is not in Q_1, the optimum algorithm makes a second query named Q_2, and if the target is not in Q_2 either, denote the third query set of optimum algorithm by Q_3, and similarly define the sequence of queries $Q_1, Q_2, \cdots, Q_w$ where Q_w is the last query.

Let a_i be the minimum index such that the sum $\sum_{m=1}^{a_i} \Pr(Q_m) \geq 1 - 1/2^i$ for any $i \geq 0$. We prove that the expected search cost of the optimum algorithm for this instance, $Opt(I(X))$ is at least $\sum_{i=1}^{\infty} Cost(\cup_{m=1}^{a_i} Q_m)/2^{i+1}$ which implies the claim of this lemma. Because $Cost(\cup_{m=1}^{a_i} Q_m)$ is cost of making a query on set $\cup_{m=1}^{a_i} Q_m$, and this set has probability at least $1 - 1/2^x$ by definition, and therefore its cost is at least $Cost_X^i$, refer to Definition 8.

In the optimum solution for each $1 \leq k \leq w$, we make query Q_k with probability $1 - \sum_{m=1}^{k-1} \Pr(Q_m)$ which is the probability that the target is not in the first $k-1$ queries. So the expected search cost of the optimum solution is at least $\sum_{m=1}^{w} Cost(Q_m) \sum_{m'=m}^{w} \Pr(Q_{m'})$ which can be rewritten as:

$$
\begin{aligned}
\sum_{i=1}^{\infty} \sum_{m=a_{i-1}+1}^{a_i} Cost(Q_m) \sum_{m'=m}^{w} \Pr(Q_{m'}) &\geq \sum_{i=1}^{\infty} \sum_{m=a_{i-1}+1}^{a_i} Cost(Q_m) \sum_{m'=a_i}^{w} \Pr(Q_{m'}) \\
&\geq \sum_{i=1}^{\infty} \sum_{m=a_{i-1}+1}^{a_i} Cost(Q_m)/2^i \\
&= \sum_{i=1}^{\infty} \left[\sum_{1}^{a_i} Cost(Q_m) - \sum_{1}^{a_{i-1}} Cost(Q_m) \right] /2^i \\
&= \sum_{i=1}^{\infty} \left[\sum_{1}^{a_i} Cost(Q_m) \right] /2^{i+1} \\
&\geq \sum_{i=1}^{\infty} Cost(\cup_{m=1}^{a_i} Q_m)/2^{i+1}
\end{aligned}
$$

The first inequality holds because m is at most a_i, and the second inequality holds because based on definition of a_i (minimality of a_i), the sum $\sum_{m'=a_i}^{w} \Pr(Q_{m'})$ should be at least $1/2^i$. The rest of the equalities and inequalities used here are straightforward.

Proof of Lemma 12

We want to prove that the nodes of tree T' can be partitioned into at most $\lceil \log(|T'|) \rceil$ sparse sets. At first we prove that there exists a sparse set $A \subseteq T'$ with $|A| \geq (|T'|+1)/2$. We prove this inductively. Let r be the root of T', and assume r has k subtrees $T'_1, T'_2, \cdots, T'_k$. By induction we know that each T'_i has a sparse subset A_i with $|A_i| \geq (|T'_i|+1)/2$, and their union $\cup_{i=1}^{k} A_i$ is a sparse set in T'. The sparse set $\cup_{i=1}^{k} A_i$ has size at least $[k + \sum_{i=1}^{k} |T'_i|]/2 = (|T'|+k-1)/2$. For $k \geq 2$, the claim holds. But if k is one, we can insert root to sparse set A_1 and have the sparse set $A_1 \cup \{r\}$ with at least $(|T'_1|+1)/2+1 = (|T'|+2)/2$, and again the claim holds. Note that in the way we find the sparse set, we do not include the root in the sparse set unless the root has one single child (subtree). In general the inductive way we take to construct the sparse set assures that no vertex v with multiple subtrees is added to our sparse set.

So there exists a sparse set A in T' with size at least $(|T'|+1)/2$. Now we find the tree T'' by removing the nodes of set A from T' as follows. We start with $T'' = T'$, and remove nodes in A one by one. Suppose v is the next vertex we want to remove. We know that based on our inductive construction every node in A including v has at most one child. If v is a leaf, we can simply remove it, and replace T'' with $T'' \setminus \{v\}$. Otherwise v has a single child w. We remove v, and make the father of v to be the new father of w. In case, v is the root (and has no father), we make w to be the new root of T''. After removing all nodes of A, we obtain the finalized tree T''.

This transition has the following property: for any pair of nodes v_1, v_2 in T'', v_1 is an ancestor of v_2 in tree T'' if and only if v_1 is an ancestor of v_2 in T'. So a sparse set in T'' is also a sparse set in T'. We can prove the claim of the lemma by induction. We know that one can partition the nodes of T'' into $\lceil \log(|T''|) \rceil$ sparse sets. These sparse sets with the sparse set A form a partition of nodes of T'. We just need to show that $\lceil \log(|T''|) \rceil + 1 = \lceil \log(2|T''|) \rceil$ is at most $\lceil \log(|T'|) \rceil$. Since $|T''|$ is $|T'| - |A|$, and A has at least $(|T'|+1)/2$ nodes, we conclude that $2|T''| \leq |T'|$ which completes the proof.

Proof of Lemma 14

Let A be this sparse set of nodes, and we want to prove that the sum of prices of nodes of A is $O(C_{OPT})$. Let $B \subseteq A$ be the set of nodes in A such that they are not descendant of any other node in A. We recall that each node of tree T is a procedure with a consecutive sequence of clusters. Assume that there are r nodes $v_1, v_2, \cdots, v_r$ in B. Let X_i be the union of clusters of node v_i, i.e. the clusters in the procedure of node v_i. We know that $X_i \cap X_j = \emptyset$ for all $1 \leq i < j \leq r$, and the probability that we reach node v_i and pay its price is exactly $\Pr(X_i)$, i.e. the sum of probabilities of objects in X_i. Let $I(X_i)$ be an instance of the problem that we are given the information that the target is in X_i, and we are supposed to find it. Note that the expected search cost of optimum algorithm might be less or higher to find the target in this instance than the original case. We denote $Opt(I(X_i))$ to be the optimum expected search cost of finding the target knowing that the target is in X_i. We have the the original optimum expected search cost, C_{OPT}, is at least $\sum_{i=1}^{r} \Pr(X_i) Opt(I(X_i))$. The latter term is the expected search cost of the optimum algorithm on the following algorithm. The instance has the same probability distribution of our original instance. We are given that the target is in which X_i, or it is in none of them. If the target is in none of the $\{X_i\}_{i=1}^{r}$, we do not need to find the target and we pay zero cost, but if it is in some X_i, we have to find the target. This is clearly an easier instance with less optimum expected search cost because we are given some free information, and in some cases we do not even need to find the target.

So it suffices to show that the total prices of node v_i and all its descendants in set A is $O(\Pr(X_i) Opt(I(X_i)))$. First of all note that all of v_i's descendants in A lie on a path from v_i to one of the leaves in its subtree because set A is sparse. So it suffices to show that the sum of prices of all nodes in a path from v_i to one of its leaves is $O(\Pr(X_i) Opt(I(X_i)))$.

Let $(v_i = u_0, u_1, u_2, \cdots, u_z)$ be this path from $v_i = u_0$ to one of the leaves in its subtree u_z. Let u_j be the first node in this path such that u_{j+1} is its left child. If there exists no such j, we define u_j to be the last node, u_z. We first bound the total prices of the nodes in the first part of the path $u_0, u_1, \cdots, u_j$, and then we focus on the remaining nodes on the path.

Assuming that the target is in subtree of node u_0, we eventually call the procedure of node u_0. In this node, we have a sequence of clusters $S_a, S_{a+1}, \cdots, S_b$. We find the largest k_0 such that $\Pr[a, k_0] < \Pr[a, b]/2$, refer to Definition 5 for definition of $\Pr[x, y]$. We query $\cup_{m=a}^{k_0} S_m$, and if the target is not there, we query set S_{k_0+1}. If we do not find it there either, we go to the right child of u_0 which is u_1. In node u_1 which is the procedure $ClusterFinder(k_0 + 2, b)$, we find the largest k_1 such that $\Pr[k_0 + 2, k_1] < \Pr[k_0 + 2, b]/2$. We similarly continue until we reach node u_j. The sum of prices up to node u_j is as follows:

$$\begin{aligned} &\Pr[a, b] C(| \cup_{m=a}^{k_0} S_m|) + \Pr[k_0 + 1, b] C(|S_{k_0+1}|) \\ &+ \sum_{x=1}^{j} \left[\Pr[k_{x-1} + 2, b] C(| \cup_{m=k_{x-1}+2}^{k_x} S_m|) + \Pr[k_x + 1, b] C(|S_{k_x+1}|) \right] \end{aligned}$$

The term $\Pr[k_{x-1} + 2, b]$ is the probability of calling procedure of node u_x, and the other probability $\Pr[k_x + 1, b]$ shows the probability that we call this procedure but we do not find the target in its first query, so we need to make the second query on the single cluster S_{k_x+1}. The first term before the summation is the price of node u_0. We will show that this sum of prices is $O(\Pr(X_i) Opt(I(X_i)))$. Equivalently we can show that the following expression which is the total prices of nodes $u_0, u_1, \cdots, u_j$ divided by $\Pr(X_i) = \Pr[a, b]$ is $O(Opt(I(X_i)))$.

$$
\begin{aligned}
\frac{\sum_{h=0}^{j} \text{price of node } u_h}{\Pr(X_i)} &= C(|\cup_{m=a}^{k_0} S_m|) + \frac{\Pr[k_0+1,b]}{\Pr[a,b]} C(|S_{k_0+1}|) + \\
&+ \sum_{x=1}^{j} \Big[\frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} C(|\cup_{m=k_{x-1}+2}^{k_x} S_m|)\Big] \\
&+ \sum_{x=1}^{j} \Big[\frac{\Pr[k_x+1,b]}{\Pr[a,b]} C(|S_{k_x+1}|)\Big] \qquad (4)
\end{aligned}
$$

Consider the instance $I(X_i)$ where we know that the target is in X_i, and we want to find it (refer to Definition 8). Using Lemma 9, we know that $Opt(I(X_i))$ is at least $\sum_{y=1}^{\infty} Cost_{X_i}^{y}/2^{y+1}$ (again refer to Definition 8 for definition of $Cost_{X_i}^{y}$). So we just need to associate the terms of expression 4 with the terms of sum $\sum_{y=1}^{\infty} Cost_{X_i}^{y}/2^{y+1}$. At first we bound the costs of the first type queries:

$$C(|\cup_{m=a}^{k_0} S_m|) + \sum_{x=1}^{j} \frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} C(|\cup_{m=k_{x-1}+2}^{k_x} S_m|)$$

We note that set $\cup_{m=a}^{k_0} S_m$ contains the objects with largest probabilities in X_i, and its probability is less than half of the total probability of X_i. So the size of set $\cup_{m=a}^{k_0} S_m$ is at most the size of set $S(X_i,1)$, and therefore its cost is upper bounded by $Cost_{X_i}^{1}$. We now consider the term $\frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} C(|\cup_{m=k_{x-1}+2}^{k_x} S_m|)$. For some integer t, we have that $1/2^{t+1} \le \frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} < 1/2^{t}$. Using the condition of the while loop in our algorithm, we can say that $\frac{\Pr[k_x+1,b]}{\Pr[a,b]}$ is greater than $1/2^{t+2}$. Because the probability of the left child of node u_x which is $\Pr[k_{x-1}+2,k_x]$ is less than half of probability of u_x which is $\Pr[k_{x-1}+2,b]$. Set $\cup_{m=a}^{k_x} S_m$ contains some objects with largest probabilities in X_i, and its probability, $1-\Pr[k_x+1,b]$, is less than $1-1/2^{t+2}$ of the total probability of X_i. This implies that the size of set $\cup_{m=a}^{k_x} S_m$ is at most the size of set $S(X_i,t+2)$, and therefore its cost is bounded by $Cost_{X_i}^{t+2}$. We can upper bound the term $\frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} C(|\cup_{m=k_{x-1}+2}^{k_x} S_m|) \le \frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]} C(|\cup_{m=a}^{k_x} S_m|)$ in the sum of prices by 8 times the term $Cost_{X_i}^{t+2}/2^{t+3}$ which is a part of the lower bound on $Opt(X_i)$.

For $0 \le j' \le j$, the portion of the price of every node $u_{j'}$ incurred by the first type query is bounded by some term in the sum $\sum_{y=1}^{\infty} Cost_{X_i}^{y}/2^{y+1}$ which is a lower bound on $Opt(X_i)$. We also know that since the probability of each child of a node in our tree is at most half of the probability of its father, we can not have two nodes $u_{j'}$ and $u_{j''}$, $0 \le j' < j'' \le j$, associated with the same t and consequently associated with the same term in this sum. So the sum of these portions of prices for nodes $u_0, u_1, \cdots, u_j$ is at most 8 times the sum $\sum_{y=1}^{\infty} Cost_{X_i}^{y}/2^{y+1}$, and consequently at most 8 times $Opt(I(X_i))$.

Now we have to bound the remaining portions of prices (inucrred by second type queries) in nodes $u_0, u_1, \cdots, u_j$ which is: $\sum_{x=0}^{j} \big[\Pr[k_x+1,b] C(|S_{k_x+1}|)\big]$. It suffices to prove that $\sum_{x=0}^{j} \big[\frac{\Pr[k_x+1,b]}{\Pr[a,b]} C(|S_{k_x+1}|)\big]$ is $O(Opt(X_i))$. For each $x \ge 0$, we have two cases: $\Pr(S_{k_x+1})$ is either at most half of $\Pr[k_x+1,b]$ or greater than that. Let $IndexHalf$ be the set of all indices like $x \ge 0$ such that: $\Pr(S_{k_x+1}) \ge \Pr[k_x+1,b]/2$. We note that since the clusters are disjoint subsets of objects, we have that:

$$Opt(X_i) \ge \sum_{0 \le x \le j | x \in IndexHalf} \Pr(S_{k_x+1}) Opt(S_{k_x+1}) \ge \sum_{0 \le x \le j | x \in IndexHalf} \Pr[k_x+1,b] Opt(S_{k_x+1})/2$$

We also know that $Opt(S_{k_x+1})$ is at least $Cost^1_{S_{k_x+1}}/2^2$ using Lemma 9. We note that since all probabilities of objects in cluster S_{k_x+1} are in range $[p, 2p]$ for some value of p, the size of $S(S_{k_x+1}, 1)$ is at least $|S_{k_x+1}|/3$. We conclude that $Opt(S_{k_x+1})$ is at least $C(|S_{k_x+1}|/3)/4 \geq C(|S_{k_x+1}|)/12$. This inequality comes from subadditivity of the cost function, in the other words, we have that $3C(|S|/3) = C(|S|/3) + C(|S|/3) + C(|S|/3) \geq C(|S|)$ for any set S. Thus we have that $\sum_{0 \leq x \leq j | x \in IndexHalf} \Pr[k_x + 1, b] Opt(S_{k_x+1})$ is at most $12 \times 2Opt(X_i) = O(Opt(X_i))$.

For the other indices, we have that $\Pr(S_{k_x+1})$ is at most $\Pr[k_x+1, b]/2$. We also know that $\Pr[k_{x-1}+2, k_x]$ is less than half of $\Pr[k_{x-1}+2, b]$. So the probability $\Pr[k_{x-1}+2, k_x+1]$ is less than $3/4$ times $\Pr[k_{x-1}+2, b]$. Similarly to the first queries, we can associate the costs of these second type queries to some terms in sum $\sum_{y=1}^{\infty} Cost^y_{X_i}/2^{y+1}$. For example, if $\frac{\Pr[k_{x-1}+2,b]}{\Pr[a,b]}$ is in range $[1/2^{t+1}, 1/2^t)$ for some t, we can associate the term $\frac{\Pr[k_x+1,b]}{\Pr[a,b]} C(|S_{k_x+1}|)$ with the term $Cost^{t+3}_{X_i}/2^{t+4}$. This way the total remaining portions of prices will be upper bounded by $O(Opt(X_i))$ as well.

Now we just need to bound the sum of prices of the remaining nodes in the path: $u_{j+1}, u_{j+2}, \cdots, u_z$. We note that all nodes $u_{j+1}, u_{j+2}, \cdots, u_z$ are in the left subtree of node u_j. So all queries we make in these nodes are just some subsets of the first query we make in node u_j. Let p be the probability that we call the procedure of node u_j, and let Q be the first query we make in node u_j. Clearly the price of u_j is at least $pC(|Q|)$. We know that the probability that we call procedure of node u_{j+1} is at most $p/2$, and for procedure of node u_{j+2}, this probability is at most $p/4$, and so on. We also know that in each of the remaining nodes, we make two queries which are both subsets of Q. So the sum of all prices of nodes $u_{j+1}, u_{j+2}, \cdots, u_z$ is at most the probability $p/2 + p/4 + \cdots = p$ times twice $C(|Q|)$. So the sum of prices of nodes $u_{j+1}, u_{j+2}, \cdots, u_z$ is at most twice the price of node u_j. We also proved that the total of prices of nodes $u_0, u_1, \cdots, u_j$ is at most 8 times $\Pr(X_i)Opt(I(X_i))$. We conclude that the total prices of nodes in the path $(u_0, u_1, \cdots, u_z)$ is $O(\Pr(X_i)Opt(I(X_i)))$. This means that the total prices of all nodes in the sparse set A is at most $O(\sum_{i=1}^{r} \Pr(X_i)Opt(I(X_i))) = O(C_{OPT})$ which concludes the claim of our lemma.

Proof of Lemma 16

We want to prove in this lemma that the expected search cost of phase 2 of Algorithm 2 incurred in cases that the target t_* is in cluster S_i is not more than $2\Pr(S_i)OPT(I_i)$. We define two special instances of a new problem to show a lower bound and an upper bound on the expected search cost of the optimal solution of the instance on cluster S_i, $OPT(I_i)$.

We adopt the original problem as follows. The probability distribution on the objects is intact. If the target is not in cluster S_i, the object containing target will be revealed in advance before we start making our queries (and therefore there is no need to make any query). So with probability $1 - \Pr(S_i)$, we make no query, and the target is found in advance. But with probability $\Pr(S_i)$, the target is in S_i, and we will be informed about this, but we still have to find the target in cluster S_i. It is clear that the optimal expected search cost is $\Pr(S_i)OPT(I_i)$ for this special problem and instance.

Now if we change this instance by reducing the probability of some object in S_i by ϵ, and increasing the probability of some object outside S_i by ϵ, the expected search cost of any strategy on this instance becomes smaller. It is intuitively clear, but here is a formal proof. Any strategy to find the target has a search tree, and in each node of the search tree we make a query on a subset of S_i. Note that if the target is outside S_i, it will be announced in advanced. For each node v in the search tree, there is a set $S'_v \subset S_i$ such that we reach node

v in the search tree if and only if the target is in S'_v. Note that by decreasing the probability of some object in S_i and increasing the probability of some object outside S_i, we do not increase the probability that the target is in $S'_v \subset S_i$ for any node v. So the expected search cost of the strategy decreases by this action.

We know that all probabilities of objects in S_i are in range $(p = 1/2^{i+1}, 2p = 1/2^i]$. Let x be the number of objects in S_i. Let P be the probability distribution on the objects. Reduce the probabilities of objects in S_i and increase the probabilities of some objects outside S_i arbitrarily to make the probability of all objects in S_i equal to p. Name this probability distribution P_L. Now increase the probabilities of objects inside S_i, and decrease some probabilities outside S_i to make all probabilities of objects in S_i equal to $2p$. Call this probability distribution P_H. Now there are six interesting measures here. We have three probability distribution, and two algorithms. One algorithm is to run the dynamic programming assuming uniform distribution on objects in S_i (we call this DynAlg), and there is optimal algorithm for distribution P, we call this OptAlg. We note that the optimum strategy is a function of probability distribution. Here we focus on the optimum strategy for distribution P, namely OptAlg. Note that in all these six problems we assume that if the object is outside S_i, it will be announced and we do not need to make any query. So we compare the expected search costs of running these two algorithms on these three distributions. As explained above, the expected search cost of a strategy increases from P_L to P, and from P to P_H. We also know that the dynamic programming is the optimal strategy on P_L and P_H. We have the following inequalities:

$$\begin{aligned} DynAlg(P_L) &\leq DynAlg(P) \leq DynAlg(P_H), \\ OptAlg(P_L) &\leq OptAlg(P) \leq OptAlg(P_H), \\ DynAlg(P_H) &\leq OptAlg(P_H). \end{aligned}$$

Since each probability of S_i in P_H is twice the corresponding probability in P_L, we have that $OptAlg(P_H) = 2OptAlg(P_L) \leq 2OptAlg(P)$. We can conclude that $DynAlg(P) \leq DynAlg(P_H) \leq OptAlg(P_H) \leq 2OptALg(P)$. So the expected search cost of the dynamic programming algorithm is at most $2\Pr(S_i)OptAlg(P) = 2\Pr(S_i)OPT(I_i)$.

Proof of Lemma 18

We note that S_{α_κ} minimizes the function f^{α_κ}. Hence, we have

$$C_{sub}(S_{\alpha_\kappa}) + \alpha_\kappa(1 - \Pr(S_{\alpha_\kappa})) \leq C_{sub}(S^*) + \alpha_\kappa(1 - \Pr(S^*)) \leq C_{sub}(S^*) + \alpha_\kappa/2.$$

Since α_κ is at most $4C_{sub}(S^*)$, we have that $C_{sub}(S_{\alpha_\kappa}) \leq 3C_{sub}(S^*)$. On the other hand, $(1 - \Pr(S_{\alpha_\kappa}))$ is at most $1/2 + C_{sub}(S^*)/\alpha_\kappa \leq 1/2 + 1/3 = 5/6$. This implies that $\Pr(S_{\alpha_\kappa})$ is at least $1/6$.