

# Compression with Graphical Constraints: An Interactive Browser

Amin Karbasi

School of Computer and Communication Sciences  
Ecole Polytechnique Fédérale de Lausanne (EPFL)  
Lausanne, Switzerland

Morteza Zadimoghaddam

Computer Science and Artificial Intelligence Laboratory (CSAIL)  
Massachusetts Institute of Technology (MIT)  
Cambridge-MA-USA

**Abstract**—We study the problem of searching for a given element in a set of objects using a membership oracle. The membership oracle, given a subset of objects  $A$ , and a target object  $t$ , determines whether  $A$  contains  $t$  or not. The goal is to find the target object with the minimum number of questions asked from the oracle. This problem is known to be strongly related to lossless source compression. In fact, the optimum strategy is provided by Huffman coding with the average number of questions very close to the entropy  $H(P)$  of the object set.

The membership oracle aims at modelling interactive methods (i.e., incorporate human feedback) has many real life applications. Due to practical constraints imposed by such applications not every subset  $A$  of objects can be queried. It is known that in general finding the optimum strategy with such constraints is NP-complete. Given this negative result we restrict attention to the cases represented by graphical models: graph  $G$  whose nodes are the database objects is given, and the queries are restricted to be those subsets  $A$  that are connected in  $G$ .

We show that when  $G$  itself is connected, there is a search algorithm that finds the target in  $4H(P) + 2$  queries on the average. Since entropy is the trivial lower bound, our algorithm performs within a constant gap from the optimum strategy.

## I. INTRODUCTION

Searching is one of the fundamental problems in computer science which arises in many different areas. In this work we consider the problem of searching where objects have different likelihood of being the target and are only accessible through a membership oracle. This is an oracle that can answer the following type of questions:

“Given a set  $A$  and an object  $t$ , does set  $A$  contain  $t$  or not?”

Given access to the membership oracle, we wish to identify a particular element (target) in the given set of objects. This element, for instance, can be a defective node in a network. In each phase, a set of objects is submitted to the oracle. Based on the outcome of the previous queries, the next query will be posed. The search terminates once the target object is located in the set. The membership oracle models, to some extent, the human abilities in content search applications with human assistance in the loop. Such applications include visual recognition [1] and pattern classification [2] where questions based on simple visual attributes are posed interactively and the goal is to identify the true class. Using this analogy, it is natural to try minimizing the access to the oracle, since asking human users is costly.

The above commonly cited examples can be seen as the visual version of the “twenty question problem” where one player chooses an instance from a category (e.g. a picture of a famous person from Flickr) and the other player who knows the category tries to guess the particular choice by posing questions as the ones specified by the membership oracle.

Without any constraints on the queries, one can see that this problem is equivalent to the lossless source compression in which Huffman coding is the optimum strategy in terms of the mean number of questions/queries asked.

However, this formulation is much too restrictive for practical content search/classification schemes. Due to limited ability of humans (e.g., imperfect memory, inexperienced users, etc) there are usually many restrictions on the subset of objects that could be asked. In practice, it might be impractical or impossible to demand a person to be capable of answering the above type of questions for every subset.

In the *constrained* twenty questions problem [3], [4] the set of possible questions is limited. The problem is then specified by two parameters: the prior distribution of objects and the set of constraints. For small databases, one can find the optimum strategy with dynamic programming while it has been shown that the general problem is NP-complete [5]. Like many hard combinatorial problems, considerable effort has been applied to finding suboptimal but still good algorithms. This is precisely what we intend to do in this work.

We address the above constraints within a general framework that we refer to as *graphical model*, i.e., each query set must conform to the constraints imposed by a graph. For instance, items can be associated with the vertices of the constraint graph and an edge between two vertices indicates that the associated pair of objects can simultaneously participate in a query. In other words, an *admissible* query is a subset of nodes that are path connected. We should stress that in this context, as a special case the optimum algorithm for finding an object on a complete graph (i.e., no graphical constraints) reduces to that of Huffman coding. Given a graphical model for the constraints we then show how one can devise a suboptimal but provably efficient algorithms with a constant gap from the performance of the optimum strategy.

The rest of this paper is organized as follows. In Section II we provide an overview of the related work in this area. In Section III, we introduce our notation and mention some basic

facts related to graphs. Then Section IV formally describes the problem that is the focus of this work, namely, compression with graphical constraints. In Section V we present our theoretical results. Finally, we conclude in Section VI.

## II. RELATED WORK

The problem studied here can be seen as a special case of the binary identification problem [3]. It is known that both the average case minimization and worse case minimization are NP-complete [5]. For both cases, there exist heuristic algorithms that admit  $O(\log n)$ -approximation [6]. There are many variants to the binary identification problem, such as searching in a totally ordered sets [7] or partially ordered sets [8], [9].

The use of interactive methods (*i.e.*, that incorporate human feedback) for searching in a dataset has a long history in literature. Relevance feedback [10] 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. Similarly, in active learning [11] the main idea is to sequentially acquire labelled data by presenting an oracle (the user) with unlabelled images and ask her to label them. Our objective in this work is somewhat similar. We use the membership oracle to identify a target in a database. This oracle has been extensively used in practice [2], [1]. In general, having access to a membership oracle is a strong assumption, since humans may not necessarily be able to answer queries of the above type for *any* object set. To make this oracle practically appealing, we introduced the graphical constraints on the set of queries, the same way deployed in [12].

Our problem is strongly related to the interactive methods used for nearest neighbour search (NNS) [13], [14] where the goal is to identify the nearest object to a query in a database. Recently, a new interactive method for the NNS was introduced by Lifshits *et al* [15] and further explored in [16], [17], [18] in which they assumed to have access to a *comparison oracle*. This oracle, given two reference objects and a query object, returns the reference object closest to the query object. Comparison oracle attempts to model another ability of humans, namely, humans are capable of comparing objects and single out which are the most similar ones, though they can probably not assign a meaningful numerical value to similarity. This problem was then generalized for the non-uniform probability distribution over the set of objects by authors in [19].

## III. DEFINITIONS AND NOTATION

In this section we introduce some tools, definition and notations which are used throughout the paper.

*Definition 1:* Consider a set of objects  $\mathcal{N}$  where  $|\mathcal{N}| = n$ . In order to represent the *constraint graph*  $G(V, E)$  we assume that there is a one-to-one map between the objects and the set of vertices  $V = \{1, 2, \dots, n\}$ . The set of edges represent the constraints on the proposed object sets as follows: we say that a subset  $A \subset V$  is *proper* if the elements of  $A$  are path connected on graph  $G$ . The proper sets are basically the ones

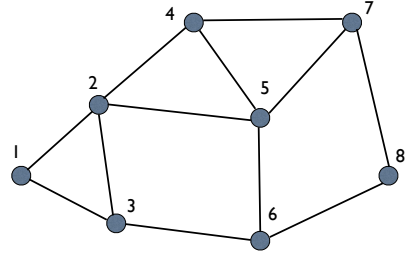


Fig. 1. The set  $A = \{2, 3, 5, 6\}$  is proper while  $B = \{1, 5, 7\}$  is not. For instance  $O_G(A, 7) = \text{NO}$  and  $O_G(A, 2) = \text{YES}$ .

that can be submitted to the membership oracle (the oracle is formally described below.) Throughout this paper we focus merely on connected constraint graphs.

*Definition 2:* The *membership oracle*  $O(A, t)$  is the one that given the set  $A \subset V$  and target object  $t$  provides one bit of information as follows:

$$O(A, t) = \begin{cases} \text{YES} & \text{if } t \in A, \\ \text{NO} & \text{otherwise.} \end{cases}$$

In the same way we can define the *constrained membership oracle* over the proper subsets  $A$ . More formally,

$$O_G(A, t) = \begin{cases} \text{YES} & \text{if } A \text{ is proper \& } t \in A, \\ \text{NO} & \text{if } A \text{ is proper \& } t \notin A, \\ \text{ERROR} & \text{if } A \text{ is not proper.} \end{cases}$$

Unless stated otherwise, throughout this paper whenever we refer to an oracle we mean the constrained membership oracle.

*Definition 3:* The *distance*  $d_G(u, v)$  in  $G$  of two vertices  $u, v$  is the length of a shortest  $u - v$  path in  $G$ . The greatest distance between any two vertices in  $G$  is the *diameter* of  $G$  and denoted by  $D_G$ .

*Definition 4:* A vertex is *central* in  $G$  if its greatest distance from any other vertex is as small as possible. This distance is the *radius* of  $G$  denoted by  $R_G$ . Formally,

$$R_G = \min_{v \in V} \max_{u \in V} d_G(u, v). \quad (1)$$

It is easy to show that  $R_G \leq D_G \leq 2R_G$ .

An example of the constraint graph  $G$  is shown in Figure 1.

*Definition 5:* For  $t \in V$ , we will call  $t$  the *target*. We will consider a probability distribution  $P = \{p_1, p_2, \dots, p_n\}$  over the set of objects which we call the *demand*. In general the demand can be heterogeneous as  $p_t$  can be different across different objects.

*Definition 6:* Let  $e = uv$  be an edge of a graph  $G(V, E)$ . By  $G/e$  we denote the graph obtained from  $G$  by *contracting* the edge  $e$  into a new vertex  $v_e$ , which becomes adjacent to all the former neighbours of  $v$  and  $u$ . The same way, for a proper set  $A \subset V$  we denote by  $G/A$ , the graph obtained by contracting all the vertices of  $A$  into one single vertex  $v_A$ . Note that if the graph  $G$  is connected, after contraction it still remains connected.

*Definition 7:* For a proper set  $A \subset G$  we denote by  $G - A$  the operation that removes all the vertices of  $A$  and their incident edges from  $G$ . Note that the *removal operation* may lead to a disconnected graph.

#### IV. PROBLEM STATEMENT

Let  $G = (V, E)$  denote the constraint graph which determines the proper sets. Given access to the constraint graph  $G$  and its corresponding membership oracle  $O_G$ , we would like to navigate through  $V$ , the set of objects, until we find the target. More formally, let  $A_k$  be the  $k$ -th proper set submitted to the constrained oracle  $O_G$ , and let

$$o_K = O_G(A_k, t) \in \{\text{YES}, \text{NO}\}$$

be the oracle's response. We define

$$H_k = \{(A_k, o_k)\}^k, \quad k = 1, 2, \dots$$

to be the *history* of the process up to the  $k$ -th access to the oracle. By definition we assume that  $H_0 = \phi$ , i.e., there is no history when we start the search. In general the selection of the  $(k+1)$ -th proposed set  $A_{k+1}$ , is determined by the history  $H_k$ . More precisely, there exists a mapping  $H_k \rightarrow f(A_k)$  such that

$$A_{k+1} = f(H_k), \quad k = 1, 2, \dots$$

We call  $f(\cdot)$  the *search policy*. Our goal is to design  $f$  such that we minimize the number of access to the oracle. More formally, given a target  $t$  and a search policy  $f$ , we define the *search cost*

$$C_f(t) = \min\{k : A_k = t\}$$

to be the number of proposals to the oracle until  $t$  is found. We restrict our attention to the case where the search policy is deterministic. The content search through membership oracle is then defined as follows:

**CONTENT SEARCH THROUGH MEMBERSHIP ORACLE (CSTMO):** Given a demand distribution  $P$  and constraint graph  $G$ , select  $f$  that minimizes the expected search cost

$$\bar{C}_f = \sum_{t \in V} p_t C_f(t).$$

#### V. MAIN RESULTS

In this section we present our main results. We first show that Algorithm 1 through contraction and removal procedures can find the target in  $H(P) + \log R_G + 2$  queries in expectation. Similarly, for Algorithm 2 we show that it identifies the target in  $\log n$  queries irrespective of the distribution or the constraint graph. Surprisingly, the combination of these two strategies will provide us with a search algorithm that performs within a constant gap from the optimum point.

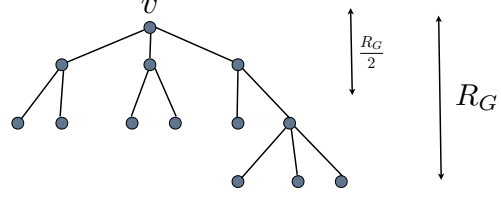


Fig. 2. The BFS spanning tree from  $v$  that shows the vertices sorted based on their distances from  $v$ . Vertex  $v$  is central, and the furthest point from it is in distance  $R_G$ .

---

#### Algorithm 1 The role of radius

---

**Input:** Constraint graph  $G$ , root vertex  $v$ , radius  $R_G$ .

**if** The constraint graph  $G$  is complete **then**  
    Run the classical Huffman Coding to find the target.  
**else**  
    Let  $S$  be the set of all objects in  $G$  with distance at most  $\lceil R_G/2 \rceil$  from  $v$ , and  $\bar{S}$  be the remaining.  
    Submit the query  $O_G(S, t)$ .  
    **if**  $O_G(S, t) = \text{YES}$  **then**  
        Recurse with new arguments:  $G - \bar{S}$ ,  $v$ , and  $\lceil R_G/2 \rceil$ .  
    **else**  
        Recurse with new arguments:  $G/S$ ,  $v_S$ , and  $\lfloor R_G/2 \rfloor$ .  
    **end if**  
**end if**

---

#### A. Divide and Conquer: The Role of Radius

Algorithm 1 divides the database into two groups and through the membership oracle determines in which group the target is located. It then reduces the search domain using contraction/removal procedure.

*Theorem 5.1:* Given a connected constraint graph  $G$  of radius  $R_G$ , Algorithm 1 finds the target in  $H(P) + \lceil \log R_G \rceil + 2$  queries, on average.

*Proof:* Unless the graph is complete Algorithm 1 divides objects into two sets: 1) Set  $S$ : the objects within distance at most  $\lceil R_G/2 \rceil$  from the central vertex  $v$ , and 2) Set  $\bar{S}$ : the remaining, i.e.,  $\bar{S} = V - S$ . The partitioning method is shown in Figure 2.

The set  $S$  is proper. Hence, using the membership oracle we can find out whether the target  $t$  is in  $S$  or not. More formally, we submit  $O_G(S, t)$ . Based on the oracle's response we do the following. If the target is in set  $S$  or equivalently  $O_G(S, t) = \text{YES}$ , we remove set  $\bar{S}$  from  $G$ , i.e.,  $G = G - \bar{S}$ . Otherwise, we contract set  $S$  since we know that the target is not in  $S$ . We need to do the contraction in order to keep the induced constraint graph connected. The above procedure is repeated (from the same central node  $v$  that we started) until the graph becomes complete, meaning, any subset of vertices can be queried. The following lemma bounds the number of access to the oracle until the first phase terminates.

*Lemma 5.2:* After at most  $1 + \lceil \log R_G \rceil$  queries, the con-

straint graph becomes complete.

*Proof:* Note that after each iteration  $R_G$  reduces to either  $\lceil R_G/2 \rceil$  or  $\lfloor R_G/2 \rfloor$ . Therefore, with at most  $\lceil \log R_G \rceil$  iterations, the  $R_G$  becomes 1. At this step, the graph is a star centered at  $v$ . We submit the query  $O_G(\{v\}, t)$ . If the target is  $v$ , the task is done, and we terminate. Otherwise, we can submit *any* subset  $S$  of leaves of this star as a proper query set by simply adding  $v$  to  $S$ . Therefore, in this case instead of submitting query  $S$ , we can submit  $S \cup \{v\}$ , and get the same answer. ■

Once the above procedure terminates we are left with a complete graph on which we can run Huffman coding for finding the target. Note that the queries we ask until the constraint graph becomes complete partitions the objects into different groups such that the constraint graph of each group is complete after contracting the set of vertices outside this group.

Let  $S_1, S_2, \dots, S_k$  denote the groups that we have at the end of the first phase. Clearly,  $\cup_{i=1}^k S_i$  is equal to  $\{1, 2, \dots, n\}$ , the set of vertices. We also know that these groups are disjoint, i.e.,  $S_i \cap S_j = \emptyset$  for every choice of  $i \neq j$ . Let  $P_i$  be the sum of probabilities of objects in group  $S_i$ . Therefore, with probability  $P_i$ , the target is in  $S_i$  on which we can then run Huffman coding to find the target. To do so, we require to access the membership oracle at most one plus the entropy of set  $S_i$ .

$$1 + \sum_{u \in S_i} \frac{p_u}{P_i} \log \left( \frac{1}{\frac{p_u}{P_i}} \right) \leq 1 + \frac{1}{P_i} \sum_{u \in S_i} p_u \log \left( \frac{1}{p_u} \right).$$

(One should not forget to scale the probability of set  $S_i$  by  $P_i$ , otherwise the sum of probabilities is not one.) The target is in  $S_i$  with probability  $P_i$ . Hence, the average number of queries in the second phase is given by

$$\sum_{i=1}^k P_i \left[ 1 + \frac{1}{P_i} \sum_{u \in S_i} p_u \log \left( \frac{1}{p_u} \right) \right] = 1 + H(P). \quad (2)$$

The target is found once both phases one and two terminate. Using Lemma 5.2 and (2) we can conclude that the average number of queries to the oracle is at most  $H(P) + \lceil \log R_G \rceil + 2$ . ■

### B. Divide and Conquer: The Role of size

In what follows we present a simple algorithm that finds the target in  $\log n$  queries.

*Theorem 5.3:* Given a connected constraint graph  $G$  with  $|V| = n$ , Algorithm 2 finds the target in  $\lceil \log n \rceil$  queries, on average.

*Proof:* First, note that the constraint graph always remains connected. In particular, if the target is in set  $T'$ , i.e.,  $O_G(T', t) = \text{YES}$ , we set  $G = T'$ . Otherwise, we contract set  $T'$  into  $v_{T'}$ , i.e.,  $G = G/T'$ . In the latter case, since the neighbors of  $v_{T'}$  are all connected through  $v_{T'}$ , whenever we want to submit a query of a subset of neighbors (which may not be path connected), we only need to add  $v_{T'}$  to the query to make the subset proper. Due to the fact that the target is

---

### Algorithm 2 The role of size

---

**Input:** Constraint graph  $G$ ,

Find a spanning tree  $T$  of the constraint graph.

**while** There are more than  $\lceil n/2 \rceil$  vertices in  $T$  **do**

    Remove one of the leaves in  $T$  arbitrarily.

**end while**

Let  $T'$  be the remaining subtree after leaf removals.

Submit the query  $O_G(T', t)$

**if**  $O_G(T', t) = \text{YES}$  **then**

    Keep all the objects in  $T'$ , and remove all the rest.

    Recurse on the remaining objects.

**else**

    Contract all objects in  $T'$ , and consider graph  $G/T'$ .

    Remove object  $v_{T'}$  from  $G/T'$ , and connect all neighbours of  $v_{T'}$ , and recurse on the remaining graph.

**end if**

---

not in set  $T'$ , this addition does not affect the response of the query. For simplicity, we can connect all neighbors of  $v_{T'}$ , keeping in mind that they are connected through  $v_{T'}$ .

Each query to the membership oracle halves the number of vertices/objects in the constraint graph. Therefore, we submit at most  $\lceil \log n \rceil$  queries. ■

The main drawback of algorithm 1 and 2 is that they can potentially deviate from the entropy  $H(P)$  by a factor of  $\log n$ .

### C. Constant Approximation Algorithm

In this section we present a search algorithm that finds the target within a constant gap from the optimum algorithm. To do so, we use algorithm 1 and 2 as the subroutines in our search strategy.

*Theorem 5.4:* Given a connected constraint graph  $G$ , there exists a search strategy that finds the target in  $2 + 4H(P)$  queries, on average.

*Proof:* Without loss of generality, let us assume that  $p_1 \geq p_2 \geq \dots \geq p_n$ . For any non-negative integer  $i$ , define the *threshold*  $a_i$  to be the number such that

$$\sum_{j: p_j < \frac{1}{2^{a_i}}} p_j \leq \frac{1}{2^i} \leq \sum_{j: p_j \leq \frac{1}{2^{a_i}}} p_j.$$

For convenience, we define  $a_0$  to be zero.

*Lemma 5.5:*  $H(P) \geq \sum_{k=1}^{\infty} a_k / 2^{k+1}$ .

*Proof:* For any  $i \geq 1$ , let  $T_i$  be the set of probabilities in the range  $(1/2^{a_i+1}, 1/2^{a_i}]$ , i.e.,

$$T_i = \{p_j | 1/2^{a_i+1} < p_j \leq 1/2^{a_i}\}.$$

We define  $Q_i$  to be the sum of probabilities in  $T_i$ , namely,  $Q_i = \sum_{p_j \in T_i} p_j$ . Clearly, for any  $p_j \in T_i$  we have  $a_i \leq \log(1/p_j)$ . As a result

$$H(P) \geq \sum_{j=1}^{\infty} Q_j a_j.$$

Note that  $\sum_{j=1}^{\infty} Q_j a_j$  can be written as

$$\sum_{j=1}^{\infty} Q_j \sum_{k=1}^j (a_k - a_{k-1}) = \sum_{k=1}^{\infty} (a_k - a_{k-1}) \sum_{j=k}^{\infty} Q_j.$$

By the way we defined  $Q_j$ , we can conclude that

$$\begin{aligned} H(P) &\geq \sum_{k=1}^{\infty} (a_k - a_{k-1}) \sum_{j: p_j \leq \frac{1}{2^{2^k}}} p_j \\ &\geq \sum_{k=1}^{\infty} (a_k - a_{k-1}) / 2^k. \end{aligned}$$

Note that the term on the r.h.s of the above expression equals  $\sum_{k=1}^{\infty} a_k / 2^{k+1}$ . ■

In Our algorithm we use thresholds  $a_i$  to partition the objects into different groups, and search inside them separately. Let  $A_i$  be the set of probabilities  $p_j$  in the range  $[1/2^{a_i}, 1]$ , i.e.  $A_i = \{p_j | p_j \geq 1/2^{a_i}\}$ . Since each probability  $p_j$  represents an object, we abuse the notation and let  $A_i$  also denote the set of corresponding objects.

*Lemma 5.6:* Given that the set  $A_i$  contains the target, Algorithm 2 finds it in  $a_i$  queries.

*Proof:* For simplicity, let us assume that  $t \in A_1$  (the argument is the same for other  $A_i$ ). In this case, we can delete all other vertices, and keep only set  $A_1$ . By doing so we might let the graph become disconnected. To avoid that, if there was is a path between two vertices  $u$  and  $v$  both in  $A_1$  such that some of the vertices of this path are outside  $A_1$ , we simply add an auxiliary edge between  $u$  and  $v$ . This way the graph stays connected. However, we should be careful about the meaning of the added edge. When we want to make a query that contains both  $u$  and  $v$ , the axillary edge indicates which vertices outside set  $A_1$  should also be included to the query. Since we know that the target is in set  $A_1$ , adding these vertices will not change the response of the oracle. We know that the probability of each vertex/object in  $A_1$  is at least  $1/2^{a_1}$ , so there are at most  $2^{a_1}$  vertices in  $A_1$ . Therefore, with at most  $a_1$  queries algorithm 2 finds the target. ■

The search algorithm starts from  $A_1$ , and uses Algorithm 2 to find some vertex  $v \in A_1$  as the target location. The search algorithm submits an additional query  $O_G(\{v\}, t)$ . If the target is  $v$ , we identify it. Otherwise, we can conclude that the target is not in  $A_1$ . The search then moves to the set  $A_2$ . In general, once we find out that the target is not in set  $A_i$  we check the subsequent set  $A_{i+1}$ . Using the same argument, with  $a_i + 1$  queries we can conclude whether  $A_i$  contains the target or not and in case it does we identify it. Note that the probability that the target is not in  $A_i$  decreases exponentially as  $i$  increases. More formally, for any  $i \geq 1$ , the probability that we do not find the target in  $A_i$  is

$$\sum_{p_j < \frac{1}{2^{a_i}}} p_j \leq \frac{1}{2^i}.$$

Hence, the expected number of queries in our algorithm is at most

$$\sum_{k=1}^{\infty} (a_k + 1) / 2^{k-1} \stackrel{(a)}{\leq} 2 + 4H(P),$$

where in (a) we used Lemma 5.5. ■

## VI. CONCLUSION

We studied the problem of content search through a database of objects using a constrained membership oracle. We showed that for those cases that the constraints can be represented by a graphical model, there exists an efficient search algorithm that finds the target object in less than  $2 + 4H(P)$  queries on the average.

## REFERENCES

- [1] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie, "Visual recognition with humans in the loop," in *ECCV (4)*, 2010, pp. 438–451.
- [2] D. Geman and B. Jedynek, "Shape recognition and twenty questions," in Proc. Reconnaissance des Formes et Intelligence Artificielle (RFIA), Tech. Rep., 1993.
- [3] M. Garey, "Optimal binary identification procedures," in *SIAM J. Appl. Math.*, 1972.
- [4] D. W. Loveland, "Performance bounds for binary testing with arbitrary weights," vol. 22, 1985, pp. 101–114.
- [5] L. Hyafil and R. Rivest, "Constructing optimal binary decision trees is np-complete," in *Information Processing Letters*, 1976, pp. 15–17.
- [6] S. R. Kosaraju, T. M. Przytycka, and R. S. Borgstrom, "On an optimal split tree problem," in *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, ser. WADS '99. Springer-Verlag, 1999, pp. 157–168.
- [7] D. Knuth, *The Art of Computer Programming*. Addison Wesley Longman Publishing Co., Inc, 1998, vol. 3.
- [8] Y. Ben-Asher, E. Farchi, and I. Newman, "Optimal search in trees," 1999.
- [9] R. Carmo, J. Donadelli, Y. Kohayakawa, and E. Laber, "Searching in random partially ordered sets," *Theor. Comput. Sci.*, vol. 321, pp. 41–57, 2004.
- [10] X. S. Zhou and T. S. Huang, "Relevance feedback in image retrieval: A comprehensive review," *Multimedia Systems*, vol. 8, no. 6, pp. 536–544, 2003. [Online]. Available: <http://dx.doi.org/10.1007/s00530-002-0070-3>
- [11] A. Holub, P. Perona, and M. Burl, "Entropy-based active learning for object recognition," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, 2008, pp. 1–8.
- [12] M. Cheraghchi, A. Karbasi, S. Mohajerzefreh, and V. Saligrama, "Graph-Constrained Group Testing," in *2010 IEEE International Symposium on Information Theory*, 2010.
- [13] K. L. Clarkson, "Nearest-neighbor searching and metric space dimensions," in *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, G. Shakhnarovich, T. Darrell, and P. Indyk, Eds. MIT Press, 2006, pp. 15–59.
- [14] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.
- [15] N. Goyal, Y. Lifshits, and H. Schutze, "Disorder inequality: a combinatorial approach to nearest neighbor search," in *WSDM*, 2008.
- [16] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *SODA*, 2009.
- [17] D. Tschopp and S. N. Diggavi, "Approximate nearest neighbor search through comparisons," 2009.
- [18] —, "Facebrowsing: Search and navigation through comparisons," in *ITA workshop*, 2010.
- [19] A. Karbasi, S. Ioannidis, and L. Massoulie, "Content search through comparisons," in *Proceedings of the 38th international colloquium conference on Automata, languages and programming*, ser. ICALP'11, 2011.