Twenty (Simple) Questions

Yuval Dagan
Technion — Israel Institute of Technology
Computer Science Department
Haifa, Israel
yuval.dagan@cs.technion.ac.il

Yuval Filmus
Technion — Israel Institute of Technology
Computer Science Department
Haifa, Israel
yuvalfi@cs.technion.ac.il

Ariel Gabizon
Zerocoin Electronic Coin Company (Zcash)
Lakewood, CO, USA
ariel.gabizon@gmail.com

Shay Moran
University of California at San Diego
Computer Science and Engineering Department
San Diego, CA, USA
Simons Institute for the Theory of Computing
Berkeley, CA, USA
shaymoran1@gmail.com

ABSTRACT

A basic combinatorial interpretation of Shannon's entropy function is via the "20 questions" game. This cooperative game is played by two players, Alice and Bob: Alice picks a distribution π over the numbers $\{1, \ldots, n\}$, and announces it to Bob. She then chooses a number x according to π, and Bob attempts to identify x using as few Yes/No queries as possible, on average.

An optimal strategy for the "20 questions" game is given by a Huffman code for π: Bob's questions reveal the codeword for x bit by bit. This strategy finds x using fewer than $H(\pi)+1$ questions on average. However, the questions asked by Bob could be arbitrary. In this paper, we investigate the following question: *Are there restricted sets of questions that match the performance of Huffman codes, either exactly or approximately?*

Our first main result shows that for every distribution π, Bob has a strategy that uses only questions of the form "$x < c$?" and "$x = c$?", and uncovers x using at most $H(\pi)+1$ questions on average, matching the performance of Huffman codes in this sense. We also give a natural set of $O(rn^{1/r})$ questions that achieve a performance of at most $H(\pi)+r$, and show that $\Omega(rn^{1/r})$ questions are required to achieve such a guarantee.

Our second main result gives a set Q of $1.25^{n+o(n)}$ questions such that for every distribution π, Bob can implement an *optimal* strategy for π using only questions from Q. We also show that $1.25^{n-o(n)}$ questions are needed, for infinitely many n. If we allow a small slack of r over the optimal strategy, then roughly $(rn)^{\Theta(1/r)}$ questions are necessary and sufficient.

STOC'17, Montreal, Canada

DOI: 10.1145/3055399.3055422

CCS CONCEPTS

• **Mathematics of computing → Combinatoric problems**; **Information theory**;

KEYWORDS

combinatorial search theory, twenty questions game, binary decision tree, information theory, redundancy

ACM Reference format:
Yuval Dagan, Yuval Filmus, Ariel Gabizon, and Shay Moran. 2017. Twenty (Simple) Questions. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 13 pages.
DOI: 10.1145/3055399.3055422

1 INTRODUCTION

A basic combinatorial and operational interpretation of Shannon's entropy function, which is often taught in introductory courses on information theory, is via the "20 questions" game (see for example the well-known textbook [17]). This game is played between two players, Alice and Bob: Alice picks a distribution π over a (finite) set of objects X, and announces it to Bob. Alice then chooses an object x according to π, and Bob attempts to identify the object using as few Yes/No queries as possible, on average.[1]

The "20 questions" game is the simplest model of *combinatorial search theory* [3] and of *combinatorial group testing* [20]. It also has a natural interpretation in the context of *interactive learning* [15]: Bob wishes to learn the secret x, and may interact with the environment (Alice) by querying features of x.

What questions should Bob ask? An optimal strategy for Bob is to compute a Huffman code for π, and then follow the corresponding decision tree: his first query, for example, asks whether x lies in the left subtree of the root. While this strategy minimizes the

[1] Another basic interpretation of Shannon's entropy function is via the transmission problem, in which Alice wishes to transmit to Bob a message x drawn from a distribution π over X, over a noiseless binary channel, using as few bits as possible on average. While the entropy captures the complexity of both problems, the two problems differ in the sense that in the "20 questions" game, Alice is "passive": she only answers the queries posed by Bob, but she does not help him by "actively" transmitting him the secret x, as she does in the transmission problem.

expected number of queries, the queries themselves could be arbitrarily complex; already for the first question, Huffman's algorithm draws from an exponentially large reservoir of potential queries.

Therefore, it is natural to consider variants of this game in which the set of queries used is restricted; for example, it is plausible that Alice and Bob would prefer to use queries that (i) can be communicated efficiently (using as few bits as possible), and (ii) can be tested efficiently (i.e. there is an efficient encoding scheme for elements of X and a fast algorithm that given $x \in X$ and a query q as input, determines whether x satisfies the query q).

We summarize this with the following meta-question, which guides this work:

Main question

Are there "nice" sets of queries such that for any distribution, there is a "high quality" strategy that uses only these queries?

Formalizing this question depends on how "nice" and "high quality" are quantified.

Variants of this question, which focus on specific sets of queries, are commonplace in computer science and related fields. Here are just a few examples:

- Binary decision trees, as used in statistics, machine learning, pattern recognition, data mining, and complexity theory, identify X as a subset of $\{0,1\}^n$, and allow only queries of the form "$x_i = 1$?".
- Feature selection and generation in machine learning can naturally be interpreted as the task of finding queries that reveal a lot of information about the learned concept.
- In the setting of binary search trees, X is a linearly ordered set, and the set of queries Q is the set of comparison queries, "$x < c$, $x = c$, or $x > c$?".[2]
- In the setting of comparison-based sorting algorithms, X is the set of permutations or linear orders over an array $x_1, \ldots, x_n$, and Q contains queries of the form "$x_i < x_j$?".
- Algebraic decision trees, used in computational geometry and studied in complexity theory, identify $X = \mathbb{F}^n$ for some field $\mathbb{F}$, and allow queries of the form "$P(\vec{x}) = 0$?" (or "$P(\vec{x}) \geq 0$?" when $\mathbb{F}$ is an ordered field) for low-degree polynomials.
- Searching in posets generalizes binary search trees by allowing X to be an arbitrary poset (usually a rooted tree). The set of queries Q is the set of comparison queries of the form "$x < c$?".

We consider two different benchmarks for sets of queries:

(1) **An information-theoretic benchmark**: A set of queries Q has *redundancy* r if for every distribution π there is a strategy using only queries from Q that finds x with at most $H(\pi) + r$ queries on average when x is drawn according to π.

(2) **A combinatorial benchmark**: A set of queries Q is r-*optimal* (or has *prolixity* r) if for every distribution π there is a strategy using queries from Q that finds x with at most $\mathrm{Opt}(\pi)+r$ queries on average when x is drawn according to π, where $\mathrm{Opt}(\pi)$ is the expected number of queries asked by an optimal strategy for π (e.g. a Huffman tree).

Given a certain redundancy or prolixity, we will be interested in sets of questions achieving that performance that (i) are as small as possible, and (ii) allow efficient construction of high quality strategies which achieve the target performance. In some cases we will have to settle for only one of these properties.

Information-theoretical benchmark. Let π be a distribution over X. A basic result in information theory is that every algorithm that reveals an unknown element x drawn according to π (in short, $x \sim \pi$) using Yes/No questions must make at least $H(\pi)$ queries on average. Moreover, there are algorithms that make at most $H(\pi)+1$ queries on average, such as Huffman coding and Shannon–Fano coding. However, these algorithms may potentially use arbitrary queries.

Are there restricted sets of queries that match the performance of $H(\pi) + 1$ queries on average, for every distribution π? Consider the setting in which X is linearly ordered (say $X = [n]$, with its natural ordering: $1 < \cdots < n$). Gilbert and Moore [34], in a result that paved the way to arithmetic coding, showed that two-way comparison queries ("$x < c$?") almost fit the bill: they achieve a performance of at most $H(\pi) + 2$ queries on average. Our first main result shows that the optimal performance of $H(\pi) + 1$ can be achieved by allowing also equality queries ("$x = c$?"):

Theorem 1.1. *For every distribution π there is a strategy that uses only comparison and equality queries which finds x drawn from π with at most $H(\pi) + 1$ queries on average. Moreover, such a strategy can be computed in time $O(n \log n)$.*

In a sense, this result gives an affirmative answer to our main question above. The set of comparison and equality queries (first suggested by Spuler [81]) arguably qualifies as "nice": linearly ordered universes appear in many natural and practical settings (numbers, dates, names, IDs) in which comparison and equality queries can be implemented efficiently. Moreover, from a communication complexity perspective, Bob can communicate a comparison/equality query using just $\log_2 n + 1$ bits (since there are just $2n$ such queries). This is an exponential improvement over the $\Omega(n)$ bits he would need to communicate had he used Huffman coding.

We extend this result to the case where X is a set of vectors of length r, $\vec{x} = (x_1, \ldots, x_r)$, by showing that there is a strategy using entry-wise comparisons ("$x_i < c$?") and entry-wise equalities ("$x_i = c$?") that achieves redundancy r:

Theorem 1.2. *Let X be the set of vectors of length r over a linearly ordered universe. For every distribution π there is a strategy that uses only entry-wise comparison queries and entry-wise equality queries and finds $\vec{x} \sim \pi$ with at most $H(\pi)+r$ queries. Moreover, this strategy can be computed in time $O(|X| \log |X|)$.*

This shows that in settings that involve lists or vectors, entry-wise comparison and equality queries achieve redundancy that is equal to the length of the vectors. The theorem is proved by applying the algorithm of the preceding theorem to uncover the vector $\vec{x}$ entry by entry.

[2] A comparison query has three possible answers: "<", "=", and ">", and so in this setting it is more natural to consider the variant of the "20 questions" game in which three answers are allowed.

As a toy example, imagine that we wish to maintain a data structure that supports an operation $find(\vec{x})$, where each x is represented by a list $(x_1, \ldots, x_r)$ of r numbers (think of r as small, say $r = 9$ and the $\vec{x}$'s denote Social Security Numbers). Moreover, assume that we have a good prior estimate on $\pi(x)$ — the frequency of $find(x)$ operations (derived from past experience, perhaps even in an online fashion). The above corollary shows how to achieve an amortized cost of $H(\pi) + r$ per $find(\vec{x})$ operation, accessing the input only via queries of the form "$x_i < c$?" and "$x_i = c$?".

As a corollary, we are able to determine almost exactly the minimum size of a set of queries that achieves redundancy $r \geq 1$. In more detail, let $u^H(n, r)$ denote the minimum size of a set of queries Q such that for every distribution π on $[n]$ there is a strategy using only queries from Q that finds x with at most $H(\pi) + r$ queries on average, when x is drawn according to π.

Corollary 1.3. *For every $n, r \in \mathbb{N}$,*

$$\frac{1}{e} r n^{1/r} \leq u^H(n, r) \leq 2 r n^{1/r}.$$

Obtaining this tight estimate of $u^H(n, r) = \Theta(r n^{1/r})$ hinges on adding equality queries; had we used only entry-wise comparison queries and the Gilbert–Moore algorithm instead, the resulting upper bound would have been $u^H(n, r) = O\left(r n^{2/r}\right)$, which is quadratically bigger.

Combinatorial benchmark. The analytical properties of the entropy function make $H(\pi)$ a standard benchmark for the average number of bits needed to describe an element x drawn from a known distribution π, and so it is natural to use it as a benchmark for the average number of queries needed to find x when it is drawn according to π. However, there is a conceptually simpler, and arguably more natural, benchmark: $\mathrm{Opt}(\pi)$ — the average number of queries that are used by a best strategy for π (several might exist), such as one generated by Huffman's algorithm.

Can the optimal performance of Huffman codes be matched *exactly*? Can it be achieved without using all possible queries? Our second main result answers this in the affirmative:

Theorem 1.4. *For every n there is a set Q of $1.25^{n+o(n)}$ queries such that for every distribution over $[n]$, there is a strategy using only queries from Q which matches the performance of the optimal (unrestricted) strategy* exactly. *Furthermore, for infinitely many n, at least $1.25^{n-o(n)}$ queries are required to achieve this feat.*

Consider a setting (similar to the one analyzed in [2, 86]) in which Alice (the client) holds an element x coming from a domain of size n, and Bob (the server) tries to determine it. Bob (who has collected statistics on Alice) knows the distribution π of x, but Alice does not. Furthermore, the downlink channel from Bob to Alice is much cheaper than the uplink channel in the opposite direction. If Bob's goal is to minimize the amount of information that Alice sends him, he can use a Huffman code to have her send the minimum amount of information on average, namely $\mathrm{Opt}(\pi)$. Naively representing a question as an arbitrary subset of $[n]$, Bob has to send $\mathrm{Opt}(\pi) \cdot n$ bits on average to Alice. Our theorem allows him to cut that amount by a factor of $\log 2 / \log 1.25 \approx 3.1$, since it only takes $\log_2 1.25^n$ bits to specify a question from Q.

One drawback of our construction is that it is randomized. Thus, we do not consider it particularly "efficient" nor "natural". It is interesting to find an explicit set Q that achieves this bound. Our best explicit construction is:

Theorem 1.5. *For every n there is an explicit set Q of $O(2^{n/2})$ queries such that for every distribution over $[n]$, there is a strategy using only queries from Q which matches the performance of the optimal (unrestricted) strategy* exactly. *Moreover, we can compute this strategy in time $O(n^2)$.*

It is natural to ask in this setting how small can a set of queries be if it is *r-optimal*; that is, if it uses at most $\mathrm{Opt}(\pi) + r$ questions on average when the secret element is drawn according to π, for small $r > 0$. Let $u^{\mathrm{Opt}}(n, r)$ denote the minimum size of a set of queries Q such that for every distribution π on $[n]$ there is a strategy using only queries from Q that finds x with at most $\mathrm{Opt}(\pi) + r$ queries on average when x is drawn from π. We show that for any fixed $r > 0$, significant savings can be achieved:

Theorem 1.6. *For all $r \in (0, 1)$:*

$$(r \cdot n)^{\frac{1}{4r}} \lesssim u^{\mathrm{Opt}}(n, r) \lesssim (r \cdot n)^{\frac{16}{r}}.$$

Instead of the exponential number of questions needed to match Huffman's algorithm exactly, for fixed $r > 0$ an r-optimal set of questions has polynomial size. In this case the upper bound is achieved by an explicit set of queries Q_r. We also present an efficient randomized algorithm for computing an r-optimal strategy that uses queries from Q_r.

Spread-out distributions. Before closing this introduction, let us go back to the information-theoretical benchmark. We mentioned that the best performance that can be achieved is $H(\pi) + 1$. This is due to distributions in which one element has probability $1 - \epsilon$: such distributions have very small entropy, but require at least one question to disambiguate the probable element form the rest.

When we rule out such distributions, by requiring all probabilities to be small, the best achievable performance improves to $H(\pi) + 0.086$, a classical result of Gallager [29]. Comparison and equality queries also benefit from such a promise:

Theorem 1.7. *For every distribution π over $[n]$ in which all elements have low probability there is a strategy that uses only comparison and equality queries and finds x drawn from π with at most $H(\pi) + 0.586$ queries on average.*

There are distributions π, in which all elements have low probability, that require $H(\pi) + 0.5011$ comparison and equality queries on average to find x drawn from π.

Comparison and equality queries thus no longer match the optimal performance in this setting, but they do take advantage of it. The algorithm used to construct the strategy mentioned in the theorem is very different from the one used to achieve $H(\pi) + 1$, and therefore may be of independent interest.

Organization of the paper. We provide a more complete summary of the paper in Section 2, followed by a brief literature review in Section 3. Appendix A corrects reference [18].

This version of the paper is an advertisement for the complete version of the paper, which is available on arXiv.

2 PAPER OUTLINE

In this section we outline our results in more detail, and give some idea about their proofs.

We start with a few formal definitions:

Definition 2.1. Let $X_n = \{x_1, \ldots, x_n\}$. We think of X_n as linearly ordered: $x_1 \prec x_2 \prec \cdots \prec x_n$.

Consider the following game: Alice picks an element $x \in X_n$, which we call the *secret element* as it is unknown to Bob, and Bob asks Yes/No (binary) questions about x, one after the other, until x is revealed. Each question is of the form "$x \in A$?" for some subset $A \subseteq X_n$.

Bob's strategy can be modeled using a *decision tree*, which is a rooted binary tree in which each internal vertex (including the root) is labeled by a *question* (a subset of X_n), the two outgoing edges are labeled *Yes* and *No*, and each leaf is labeled by an element of X_n. Such a tree corresponds to a strategy for Bob: he starts by asking the question at the root, and continues recursively to one of its children according to the answer. When reaching a leaf, the secret element is revealed to be the leaf's label. Thus, in order for a tree to be valid, the label of each leaf must be the only element in X_n which is consistent with the answers to all questions on the way. If a decision tree uses only questions from $\mathcal{Q} \subseteq 2^{X_n}$, we call it a decision tree *using* $\mathcal{Q}$.

Alice picks the element x from a distribution $\pi = (\pi_1, \ldots, \pi_n)$, where $\pi_i = \Pr_{x\sim\pi}[x = x_i]$. Bob, who knows π, and has to identify x among all elements in the support of π. Thus, a *decision tree for* π is defined as a decision tree for $\mathrm{supp}(\pi)$. The *cost* of such a tree is the average number of questions asked on a random element $x \sim \pi$. We denote the cost of the optimal unrestricted decision tree for π by $\mathrm{Opt}(\pi)$.

Given a distribution π, its binary entropy is defined by $H(\pi) = \sum_{i=1}^{n} \pi_i \log_2 \frac{1}{\pi_i}$. The binary entropy function is denoted $h(p) = p \log_2 \frac{1}{p} + (1-p)\log_2 \frac{1}{1-p}$ for $0 \le p \le 1$. A classical inequality due to Shannon [76] states that

$$H(\pi) \le \mathrm{Opt}(\pi) < H(\pi) + 1.$$

We say that a set of questions $\mathcal{Q}$ has *redundancy* r if for every distribution π there exists a decision tree using $\mathcal{Q}$ whose cost is at most $H(\pi)+r$. We say that it has *prolixity* r if for every distribution π there exists a decision tree using $\mathcal{Q}$ whose cost is at most $\mathrm{Opt}(\pi)+r$.

We will be particularly interested in the following sets of questions:

- Equality queries: questions of the form "$x = x_i$?" for $i \in \{1, \ldots, n\}$.
- Comparison queries: questions of the form "$x \prec x_i$?" for $i \in \{1, \ldots, n\}$.
- Interval queries: questions of the form "$x_i \le x \le x_j$?" for $1 \le i \le j \le n$.

Section organization. Our results on comparison and equality queries are described in Subsection 2.1. Our results on sets of questions that match the performance of Huffman codes exactly are described in Subsection 2.2. Our results on sets of question that achieve small prolixity are described in Subsection 2.3.

2.1 Comparison And Equality Queries

2.1.1 Achieving Redundancy 1. Huffman codes achieve redundancy 1. Are there small sets of questions which also achieve redundancy 1? A natural candidate is the set of comparison queries, which gives rise to a class of decision trees known as *alphabetic trees* (for the difference between these and binary search trees, see Section 3.1.2). However, for some distributions comparison queries cannot achieve redundancy smaller than 2: the distribution $(\epsilon, 1-2\epsilon, \epsilon)$ requires two comparison queries to identify the central element, but has entropy $O(\epsilon \log(1/\epsilon))$, which goes to 0 with ϵ.

The only way to achieve redundancy 1 on this kind of distribution is to also allow equality queries. The resulting class of decision trees, first suggested by Spuler [81, 82], is known as *two-way comparison search trees* or *binary comparison search trees*. To the best of our knowledge, the redundancy of two-way comparison search trees has not been considered before. We show:

Theorem 2.2. *Comparison and equality queries achieve redundancy* 1.

Our proof is constructive: given a distribution π, we show how to construct efficiently a decision tree using comparison and equality queries whose cost is at most $H(\pi) + 1$. Our construction is based on the weight balancing algorithm of Rissanen [73], which uses only comparison queries:

> **Weight balancing algorithm**
> Given a probability distribution π, ask a comparison query minimizing $|\Pr[\text{Yes}] - \Pr[\text{No}]|$, and recurse on the distribution of π conditioned on the answer. Stop when the secret element is revealed.

Horibe [39] showed that this algorithm achieves redundancy 2. Given the additional power of asking equality queries, it is natural to ask such queries when some element has large probability. Indeed, this is exactly what our algorithm does:

> **Weight balancing algorithm with equality queries**
> If the most probable element has probability at least 0.3, compare it to the secret element, and recurse if the answer is negative.
>
> Otherwise, ask a comparison query minimizing $|\Pr[\text{Yes}] - \Pr[\text{No}]|$, and recurse on the answer.

The constant 0.3 here is just one of the possible values for which this algorithm achieves redundancy 1.

Our analysis of the redundancy in the proof of Theorem 2.2 depends on $\pi_{\max}$, the probability of the most probable element in the distribution π. It proceeds by first showing that if $\pi_{\max} \ge 0.3$ then the redundancy is at most 1, and then recursively reducing the case in which $\pi_{\max} < 0.3$ to the case $\pi_{\max} \ge 0.3$ via a careful analysis of how $\pi_{\max}$ (that now refers to the most probable element in the recursively defined conditional distribution) varies according to the answers that the algorithm receives.

2.1.2 Higher redundancy. The weight balancing algorithms assume that the domain of π is linearly ordered. Another natural setting is when the domain is Y^r, where Y is a linearly ordered set

of size $n^{1/r}$. In other words, each element of X is a vector of length r, and the secret element is a vector $\vec{x} = (x^{(1)}, \ldots, x^{(r)})$.

A corollary to Theorem 2.2 is that in this setting, the set of all queries of the form "Is $x^{(i)}$ at most y?" and "Does $x^{(i)}$ equal y?", for $1 \le i \le r$ and $y \in Y$, achieves redundancy at most r.

Indeed, suppose that x is drawn according to a probability distribution π with components $\pi^{(1)}, \ldots, \pi^{(r)}$. If we determine the components $x^{(1)}, \ldots, x^{(r)}$ consecutively using the weight balancing algorithm with equality queries, we obtain a decision tree whose cost is at most

$$[H(\pi^{(1)}) + 1] + [H(\pi^{(2)}|\pi^{(1)}) + 1] + \cdots + [H(\pi^{(r)}|\pi^{(1)}, \ldots, \pi^{(r-1)}) + 1] = H(\pi) + r.$$

Since the number of questions is roughly $rn^{1/r}$, this proves the first part of the following theorem:

Theorem 2.3. *There is a set of $O(rn^{1/r})$ questions that achieve redundancy r.*

Moreover, $\Omega(rn^{1/r})$ questions are required to achieve redundancy r.

The other direction is proved by considering distributions almost concentrated on a single element: a set of questions Q can have redundancy r only if it identifies any single element using at most r questions, and this shows that $2^r \binom{|Q|}{\le r} \ge n$, implying the lower bound in the theorem.

We slightly improve this lower bound via a connection with *witness codes*, which are sets of vectors in $\{0,1\}^m$ in which each vector is identifiable using at most r coordinates.

2.1.3 Spread-out distributions. Theorem 2.2 shows that comparison and equality queries achieve redundancy 1. Although this is the best redundancy achievable, the only distributions which require this redundancy are those which are almost entirely concentrated on a single element. Can we obtain a better bound if this is not the case? More concretely, what redundancy is achievable when the distribution has high *min-entropy*, that is, when all elements have small probability?

To make this question precise, let r_p be the maximal redundancy of comparison and equality queries on distributions in which the maximal probability is at most p (on an arbitrarily large domain), and let $r_0 = \lim_{p \to 0} r_p$. We show:

Theorem 2.4. *The quantity r_0 is bounded by*

$$0.5011 < r_0 < 0.586.$$

The lower bound is attained by distributions of roughly the form $\epsilon, 1/n, \epsilon, 1/n, \ldots, \epsilon, 1/n, \epsilon$ for $n \approx \frac{3}{\log_2 e} \cdot 2^k$, where k is a large integer. The upper bound is proved by modifying another algorithm achieving redundancy 2 using only comparison queries: the Gilbert–Moore algorithm [34], also known as Shannon–Fano–Elias encoding, which forms the basis of arithmetic encoding. Before describing our upper bound, we first explain this algorithm and why it achieves redundancy 2.

Given a distribution π on X_n in which x_i has probability π_i, the Gilbert–Moore algorithm proceeds as follows:

Gilbert–Moore algorithm

Partition $[0,1]$ into segments $T_1, \ldots, T_n$ of lengths $\pi_1, \ldots, \pi_n$, where T_{i+1} is placed to the right of T_i. Put a point p_i at the middle of T_i for all i. Intuitively speaking, we now perform binary search on $[0,1]$ to reveal the point p_i corresponding to the secret element.

Formally, we maintain an interval $I = [a,b]$, initialized at $[0,1]$. At every iteration we ask whether the secret element resides to the left or to the right of $\frac{a+b}{2}$, and update I accordingly, decreasing its length by a half (exactly). We stop once I contains only one point.

The secret element x_i is isolated after at most $\lceil \log_2 \frac{2}{\pi_i} \rceil < \log_2 \frac{1}{\pi_i} + 2$ steps, and so the Gilbert–Moore algorithm has redundancy 2. Indeed, after $\lceil \log_2 \frac{2}{\pi_i} \rceil$ steps, the binary search focuses on an interval containing p_i whose length is at most $\pi_i/2$ and is thus contained entirely inside T_i.

The choice of π_i as the length of T_i isn't optimal. A better choice is $2^{-\ell_i}$, where ℓ_i is the length of the codeword for x_i in a Huffman code for π. The same argument as before shows that x_i is isolated after at most $\ell_i + 1$ steps, and so the modified algorithm has cost at most $\mathrm{Opt}(\pi) + 1 < H(\pi) + 2$. This algorithm appears in Nakatsu [69].

How can we use equality queries to improve on this algorithm? As in the modified weight balancing algorithm, the idea is to use equality queries to handle elements whose weight is large. However, the exact mechanism is rather different:

Random placement algorithm with interval queries

Partition $[0,1]$ into segments $T_1, \ldots, T_n$ of length $2^{-\ell_1}, \ldots, 2^{-\ell_n}$, where $\ell_1, \ldots, \ell_n$ are defined as explained above, and put a point p_i at the middle of T_i. Then rotate the entire setup by a random amount.

Perform "modified binary search" on $[0,1]$:

- If the current interval J contains a segment T_i of size $|J|/2$ in its entirety, ask whether $x = x_i$, and if the answer is negative, remove T_i from J.
- Otherwise, perform one step of binary search on J, as described in the Gilbert–Moore algorithm.

In both cases, the size of J exactly halves at each iteration of the modified binary search. The same argument as before shows that x_i is always isolated after at most $\ell_i + 1$ steps. After $\ell_i - 1$ steps, J is an interval of length $2 \cdot 2^{-\ell_i} = 2|T_i|$ containing p_i, and it contains all of T_i with probability $1/2$; in this case, x_i is isolated after only ℓ_i steps. The resulting algorithm has cost at most $\mathrm{Opt}(\pi) + 1/2$. However, since we rotated the entire setup, interval queries are needed to implement the binary search.

In order to obtain an algorithm which uses comparison queries rather than interval queries (in addition to equality queries), we need to constrain the rotation somehow. Reflecting on the argument above, we see that if all segments have length at most $2^{-\ell}$ then it suffices to randomly "slide" the setup across an interval of free space of length $2 \cdot 2^{-\ell}$. This leads to the following algorithm, in which $\ell = \min(\ell_1, \ldots, \ell_n)$:

Random placement algorithm with comparison queries

Partition $[0,1]$ into segments $T_1,\dots,T_n$ of length $2^{-\ell_1},\dots,2^{-\ell_n}$. Choose a random set of intervals of total length $4\cdot 2^{-\ell}$, and halve their lengths while keeping all intervals adjacent to each other, thus forming a slack of length $2\cdot 2^{-\ell}$ at the right end of $[0,1]$. Put a point p_i at the middle of I_i, and shift the entire setup to the right by a random amount chosen from $[0, 2\cdot 2^{-\ell}]$.

Perform modified binary search as in the preceding algorithm.

Essentially the same analysis as before shows that this algorithm, which now uses only comparison and equality queries, has cost at most $\mathrm{Opt}(\pi) + 1/2 + 4\cdot 2^{-\ell}$.

As the maximal probability $\pi_{\max}$ tends to zero, the length ℓ of the minimal codeword tends to infinity, and thus the cost is bounded by $\mathrm{Opt}(\pi) + 1/2$ in the limit. A classical result of Gallager [29] states that $\mathrm{Opt}(\pi) \le H(\pi) + 0.0861 + \pi_{\max}$, and so we obtain the bound $r_0 < 0.586$.

2.2 Optimal Sets Of Questions Of Minimal Size

We have shown above that comparison and equality queries suffice to obtain redundancy 1, matching the performance of Huffman's algorithm in this regard. What if we want to match the performance of Huffman's algorithm *exactly*?

Definition 2.5. A set of questions $\mathcal{Q}$ over X_n is *optimal* if for every distribution π there exists a decision tree using $\mathcal{Q}$ whose cost is $\mathrm{Opt}(\pi)$. Stated succinctly, a set of questions is optimal if it has zero prolixity.

It is not clear at the outset what condition can guarantee that a set of questions is optimal, since there are infinitely many distributions to handle. Fortunately, it turns out that there exists a "0-net" for this:

Definition 2.6. A distribution is *dyadic* if every non-zero element has probability $2^{-\ell}$ for some integer ℓ.

Observation 1. A set of questions is optimal for all distributions over X_n if and only if it is optimal for all *dyadic* distributions over X_n.

Roughly speaking, this observation follows from the fact that binary decision trees correspond to dyadic distributions, a property that is exploited in Section 2.1.3 as well.

A further simplification is:

Observation 2. A set of questions is optimal if and only if it is a *dyadic hitter*: for every non-constant dyadic distribution π it contains a question Q *splitting* π, that is, $\pi(Q) = \pi(\overline{Q}) = 1/2$.

This observation follows from the following characterization of optimal decision trees for dyadic distributions, whose proof is a straightforward application of the chain rule for entropy:

Lemma 2.7. *Let π be a dyadic distribution. A decision tree T for π has optimal cost* $\mathrm{Opt}(\pi)$ *if and only if the following holds for every internal node v with children v_1, v_2:*

$$\pi(v_1) = \pi(v_2) = \frac{\pi(v)}{2},$$

where $\pi(v)$ is the probability that the decision tree reaches the node v.

Our task is thus reduced to determining the size of a dyadic hitter. To understand what we are up against, consider distributions that have $2\beta n - 1$ "heavy" elements of probability $\frac{1}{2\beta n}$ each, and $(1-2\beta)n + 1$ "light" elements of total probability $\frac{1}{2\beta n}$, where β is a positive number such that βn is a power of 2.

A short argument shows that the only sets splitting such a distribution are those containing exactly βn heavy elements, and their complements. By considering all possible partitions of X_n into heavy and light elements, we see that every dyadic hitter must contain at least this many questions:

$$\begin{aligned}&\frac{\#\text{partitions to heavy and light elements}}{\#\text{partitions split by a question of size } \beta n}\\ =&\frac{\#\text{questions of size } \beta n}{\#\text{questions of size } \beta n \text{ splitting each partition}}\\ =&\frac{\binom{n}{\beta n}}{\binom{2\beta n-1}{\beta n}} \approx 2^{(h(\beta)-2\beta)n}.\end{aligned}$$

(To see the first equality, cross-multiply to get two different expressions for the number of pairs consisting of a partition to heavy and light elements and a question of size βn splitting it.)

Choosing $\beta = 1/5$ (which maximizes $h(\beta) - 2\beta$), we find out that roughly 1.25^n questions are necessary just to handle distributions of this form. There is a fine print: the value $\beta = 1/5$ is only achievable when $n/5$ is a power of 2, and so the bound obtained for other values of n is lower.

Somewhat surprisingly, there is a matching upper bound: there is a set of roughly 1.25^n questions which forms a dyadic hitter! In order to show this, we first show that for every non-constant dyadic distribution π there is some question size $1 \le c \le n$ such that

$$\frac{\#\text{questions of size } c \text{ that split } \pi}{\#\text{questions of size } c} \ge 1.25^{-n-o(n)}.$$

Therefore, by choosing $1.25^{n+o(n)}$ random questions of size c, it is highly probable that one of them splits π.

This suggests considering a random set of questions $\mathcal{Q}$ formed by taking $1.25^{n+o(n)}$ questions at random of size c for each $1 \le c \le n$. Since there are only exponentially many dyadic distributions, a union bound shows that with high probability, $\mathcal{Q}$ splits *all* dyadic distributions.

How do we identify a value of c and a large set of questions of size c splitting an arbitrary dyadic distribution π? Roughly speaking, we bucket the elements of μ according to their probabilities. Suppose that there are m non-empty buckets, of sizes $c_1,\dots,c_m$, respectively. Assume for simplicity that all c_i are even; in the actual proof, we essentially reduce to this case by bundling together elements of lower probability. A question containing exactly half the elements

in each bucket thus splits μ. The number of such questions is

$$\prod_{i=1}^{m} \binom{c_i}{c_i/2} \approx \prod_{i=1}^{m} \frac{2^{c_i}}{\sqrt{c_i}} \approx \frac{2^{c_1+\cdots+c_m}}{n^{m/2}}.$$

By "throwing out" all elements of small probability (corresponding to the light elements considered above), we can guarantee that $m \leq \log n$, ensuring that the factor $n^{-m/2}$ is subexponential; however, this means that not all elements are going to belong to a bucket. If $c_1 + \cdots + c_m = 2\beta n$ after throwing out all light elements then we have constructed roughly $2^{2\beta n}$ questions of size $c = \beta n$. Since

$$\frac{\binom{n}{c}}{2^{2\beta n}} \approx 2^{(h(\beta)-2\beta)n} \leq 1.25^n,$$

we have fulfilled our promise.

To summarize:

Theorem 2.8. *For every n there is an optimal set of questions of size $1.25^{n+o(n)}$.*

For infinitely many n, every optimal set of questions contains at least $1.25^{n-o(n)}$ questions.

While our upper bound works for every n, our lower bound only works for infinitely many n. For arbitrary n our best lower bound is only $1.232^{n-o(n)}$. We suspect that the true answer depends on the fractional part of $\log_2 n$.

Our upper bound is non-constructive: it does not give an explicit optimal set of questions of size $1.25^{n+o(n)}$, but only proves that one exists. It is an interesting open question to obtain an explicit such set of this size. However, we are able to construct an explicit optimal set of questions of size $O(\sqrt{2}^n)$:

Theorem 2.9. *For every n there is an explicit optimal set of questions $\mathcal{Q}$ of size $O(2^{n/2})$.*

Furthermore, given a distribution on X_n, we can construct an optimal decision tree using $\mathcal{Q}$ in time $O(n^2)$.

The explicit set of questions is easy to describe: it consists of all subsets and all supersets of $S = \{x_1, \ldots, x_{\lfloor n/2 \rfloor}\}$. Why does this work? Given the observations above, it suffices to show that this collection contains a question splitting any non-constant dyadic distribution π on X_n. If $\pi(S) = 1/2$ then this is clear. If $\pi(S) > 1/2$, suppose for simplicity that $\pi_1 \geq \cdots \geq \pi_{\lfloor n/2 \rfloor}$. A simple parity argument shows that $\pi(\{x_1, \ldots, x_m\}) = 1/2$ for some $m < \lfloor n/2 \rfloor$. The case $\pi(S) < 1/2$ is similar.

This set of questions, known as a *cone*, appears in the work of Lonc and Rival [60] on *fibres*, which are hitting sets for maximal antichains. The lower bound $1.25^{n-o(n)}$ on optimal sets of questions appears, in the context of fibres, in Duffus, Sands and Winkler [22]. The connection between these results and ours is that every fibre is an optimal set of questions.

2.3 Sets Of Questions With Low Prolixity

The hard distributions described in the preceding section show that any strictly optimal set of questions must have exponential size. The following theorem shows that this exponential barrier can be overcome by allowing a small prolixity:

Theorem 2.10. *For every n and $r \in (0,1)$ there is a set $\mathcal{Q}_r$ of roughly $(r \cdot n)^{16/r}$ questions which has prolixity r.*

Furthermore, at least roughly $(r \cdot n)^{0.25/r}$ questions are required to achieve prolixity r.

As in the case of prolixity 0, the lower bound relies on a family of hard distributions: Assume that r is of the form 2^{-k} for some integer $k > 0$. A hard distribution consists of $2^k - 1$ "heavy elements" of total probability $1 - \delta$, and $n - (2^k - 1)$ "light elements" of total probability δ, where $\delta = r^2/2$ (or any smaller positive number). A case analysis shows that every 2^{-k}-optimal decision tree (one whose cost exceeds the optimal Huffman cost by at most 2^{-k}) for this distribution must partition the heavy elements evenly (into parts of size 2^{k-1} and $2^{k-1} - 1$), and put all light elements in the same part. Ignoring the difference between 2^{k-1} and $2^{k-1} - 1$, this shows that any 2^{-k}-optimal set of questions must contain at least this many questions:

$$\frac{\binom{n}{2^k-1}}{\binom{n-(2^{k-1}-1)}{2^{k-1}}} \geq \left(\frac{n}{2^k}\right)^{2^{k-1}-1}.$$

In terms of $r = 2^{-k}$, this lower bound is $(r \cdot n)^{0.5/r-1}$, from which we obtain the form above (approximating an arbitrary r by a negative power of 2).

The upper bound is more involved. The set of questions consists of all interval queries with up to 2^k elements added or removed (in total, about $n^2\binom{n}{\leq 2^k}2^{2^k} = n^2 \cdot O(n/2^k)^{2^k}$ questions); we will aim at a redundancy of roughly $4 \cdot 2^{-k}$. The algorithm has some resemblance to the Gilbert–Moore line of algorithms described in Section 2.1.3. As in that section, given a distribution π, our starting point is the distribution $2^{-\ell_1}, \ldots, 2^{-\ell_n}$ formed from a Huffman code for π, where ℓ_i is the length of the codeword corresponding to x_i.

In contrast to Gilbert–Moore-style algorithms, though, instead of maintaining an interval we will maintain a *dyadic subestimate* for the probabilities of all elements "in play". That is, for every element consistent with the answers so far we will assign a probability $q_i = 2^{-t_i}$, ensuring that $\sum_i q_i \leq 1$.

The algorithm is recursive, getting as input a dyadic subdistribution $q_1, \ldots, q_m$, which is initially $2^{-\ell_1}, \ldots, 2^{-\ell_n}$; the recursion stops when $m = 1$. The algorithm classifies its input elements according to the magnitude of q_i as either *heavy* (at least 2^{-k}), or *light* (otherwise), and proceeds as follows:

Random window algorithm

Case 1: the total mass of heavy elements is at least $1/2$. Find a set of heavy elements whose probability is exactly $1/2$, and ask whether the secret element lies in that set. Double the probability of all conforming elements, and recurse with the set of conforming elements.

Case 2: the total mass of heavy elements and the total mass of light elements are at most $1/2$. Ask whether the secret element is heavy or light. Double the probability of all conforming elements, and recurse with the set of conforming elements.

Case 3: The total mass σ of light elements is at least $1/2$. Identify a light element x_i with a segment T_i of length

q_i, and place all such segments consecutively on a circle of circumference σ. Choose a random arc W of length $1/2$ (the *window*). Ask whether the secret element is a light element whose segment's midpoint lies in the window. Double the probability of all conforming elements not intersecting the boundaries of the window, and recurse with the set of conforming elements.

Achieving an optimal cost requires that at each step the probability of each conforming element is doubled. However, by Theorem 2.8 this would require exponentially many potential queries (rather than just our modified interval queries), and so we have to compromise by not doubling some of the probabilities of conforming elements in some steps (in Case 3, the probabilities of the two boundary elements are not doubled). Nevertheless, the above randomized algorithm ensures that the expected number of times each element is being "compromised" is $O(2^{-k})$.

How many questions does it take to find an element x_i whose starting probability is $2^{-\ell_i}$? At any iteration in which $q_i \geq 2^{-k}$, the probability q_i always doubles. When $q_i < 2^{-k}$, Case 3 could happen, but even then the probability that q_i doesn't double is only $2q_i/\sigma \leq 4q_i$. A straightforward calculation shows that q_i doubles after at most $\frac{1}{1-4q_i}$ questions in expectation when $q_i < 2^{-k}$, and so the expected number of questions needed to discover x_i is at most

$$k + \sum_{j=k+1}^{\ell_i} \frac{1}{1 - 4 \cdot 2^{-j}} < \ell_i + 4 \cdot 2^{-k} + \frac{2}{3}(4 \cdot 2^{-k})^2.$$

Roughly speaking, the resulting prolixity is $r \approx 4 \cdot 2^{-k}$, and the number of questions is about $n^2 \cdot O(n/2^k)^{2^k} \approx n^2 \cdot O(r \cdot n)^{4/r}$.

3 RELATED WORK

Our work can be seen as answering two different, but related, questions:

(1) How good are certain particular sets of questions (particularly comparison and equality queries), compared to the information-theoretical benchmark and the combinatorial benchmark?
(2) What is the smallest set of questions which matches the performance of Huffman codes, in terms of the information-theoretical benchmark or the combinatorial benchmark? What if we allow a small slack?

To the best of our knowledge, existing literature only deals with the first question. Apart from unrestricted questions, the only set of questions which has been extensively studied from the perspective of our two benchmarks is comparison queries.

We assume familiarity with the basic definitions appearing in Section 2.

Section organization. We describe several sets of questions which have been studied in the literature in Section 3.1. Other relevant topics are discussed in Section 3.2.

3.1 Types Of Decision Trees

We can identify sets of questions with decision trees using them (that is, using only questions from the set). For example, binary search trees (more accurately, alphabetic trees; see below) are decision trees using comparison queries.

The cost function that we study and attempt to minimize in this paper is *distributional* or *average case*: it is the average number of questions asked on an element chosen from a given distribution.

Another cost function studied in the literature, especially in problems whose motivation is algorithmic, is *worst case*: the maximum number of questions asked on any element. The most familiar examples are binary search and sorting.

We go on to list several types of decision trees appearing in the literature, briefly commenting on each of them.

3.1.1 Unrestricted decision trees. The simplest type of decision trees is unrestricted decision trees. Huffman [44] showed how to construct optimal decision trees, and van Leeuwen [83] showed how to implement his algorithm in time $O(n \log n)$, or $O(n)$ if the probabilities are sorted. Huffman also considered non-binary decision trees, generalizing his algorithm accordingly.

Gallager [29] showed that decision trees constructed by Huffman's algorithm are characterized by the *sibling property*: the nodes of the tree can be arranged in non-decreasing order of probability of being reached, in such a way that any two adjacent nodes are siblings. This shows, among else, that in some cases there are optimal decision trees which cannot be generated by Huffman's algorithm. In the parlance of codes, a distinction should be made between *Huffman codes* and the more general *minimum redundancy codes*.

Gallager also discussed the *redundancy* of Huffman codes in terms of the maximum probability of an element, showing that if π is a distribution with maximum probability $\pi_{\max}$ then $\mathrm{Opt}(\pi) \leq H(\pi) + \pi_{\max} + 1 - \log_2 e + \log_2 \log_2 e$, where $1 - \log_2 e + \log_2 \log_2 e \approx 0.086$. He also showed that the constant $1 - \log_2 e + \log_2 \log_2 e$ is optimal.

The question of the redundancy of Huffman codes in terms of $\pi_{\max}$ has attracted some attention: a line of work, including [12, 46], culminated in the work of Montgomery and Abrahams [65], who found the optimal lower bound on $H(\pi) - \mathrm{Opt}(\pi)$ in terms of $\pi_{\max}$, and in the work of Manstetten [61], who found the optimal upper bound.

The redundancy of Huffman codes has also been considered in terms of other parameters, such as the minimum probability, both minimum and maximum probabilities, or some probability. See Mohajer et al. [64] for a representative example with many references.

Gallager also gave an algorithm for dynamically updating Huffman trees, contemporaneously with Faller [26] and Knuth [53]. Their work was improved by Vitter [84]. Vitter's data structure is given an online access to a stream of symbols $(\sigma_n)_{n \in \mathbb{N}}$, and it maintains a decision tree T_n which at time n is a Huffman tree for the empirical distribution μ_n of $\sigma_1, \ldots, \sigma_n$. The update time is $O(T_n(\sigma_n))$, where $T_n(\sigma_n)$ is the depth (or codeword length) of σ_n

in T_n, and the output codewords satisfy

$$\frac{1}{n}\sum_{i=1}^{n} T_i(\sigma_i) \le \mathrm{Opt}(\mu_n) + 1.$$

Many variants of Huffman coding have been considered: length-restricted codes [24, 33], other cost functionals [8], unequal letter costs [49], and many more. See the excellent survey of Abrahams [1].

3.1.2 Binary search trees. A binary search tree (BST) stores an ordered list of elements $x_1 \prec \cdots \prec x_n$ over a linearly ordered domain, and supports a search operation, which given an element x can result in any of the following outcomes:

- $x = x_i$ for some i.
- $x \prec x_1$.
- $x_i \prec x \prec x_{i+1}$ for some $i < n$.
- $x \succ x_n$.

Each node of the tree contains an element x_i, and it tests whether $x \prec x_i$, $x = x_i$, or $x \succ x_i$. Given probabilities $p_1, \ldots, p_n$ for successful searches ($x = x_i$) and $q_0, \ldots, q_n$ for unsuccessful searches ($x_i \prec x \prec x_{i+1}$), an optimal BST is one that minimizes the number of questions it takes to identify the class of a secret element x.

Binary search trees do not fit the model considered in the paper as stated. However, if the probability of a successful search is zero (that is, $p_1 = \cdots = p_n = 0$), then the ternary queries become binary queries, and the resulting decision tree is a decision tree using comparison queries on the domain composed of the $n + 1$ *gaps* between and around the elements $x_1, \ldots, x_n$:

$$y_0 = (-\infty, x_1), y_1 = (x_1, x_2), \ldots, y_{n-1} = (x_{n-1}, x_n), y_n = (x_n, \infty).$$

The resulting model is also known as *alphabetical trees* or *lexicographical trees*, and has been suggested by Gilbert and Moore [34] in the context of variable-length binary encodings. Alphabetical trees (of words) are decision trees in which the leaves are ordered alphabetically. In our terminology, they are decision trees using comparison queries.

Kraft's inequality states that a decision tree whose leaves have depths $\ell_1, \ldots, \ell_n$ exists if and only if $\sum_{i=1}^{n} 2^{-\ell_i} \le 1$. Nakatsu [69] gave an analog of Kraft's inequality for alphabetical trees.

Knuth [52] gave an $O(n^2)$ dynamic programming algorithm for finding the optimal BST. Hu and Tucker [41] and Garsia and Wachs [32] gave $O(n \log n)$ algorithms for finding optimal alphabetical trees. These algorithms are more complicated than Huffman's.

Several heuristics for constructing good alphabetical trees are described in the literature. Gilbert and Moore [34] gave one heuristic which produces a tree with cost at most $H(\pi) + 2$. Nakatsu [69] gave the stronger bound $\mathrm{Opt}(\pi) + 1$ for a very similar heuristic.

Another heuristic, *weight balancing*, was suggested by Rissanen [73] (and even earlier by Walker and Gottlieb [85]), who showed that it produces a tree with cost at most $H(\pi) + 3$. Horibe [39] improved the analysis, showing that the cost is at most $H(\pi) + 2$.

The question of the redundancy of alphabetical trees has been considered in the literature. While the bound $H(\pi)+2$ cannot be improved upon in general, a better bound can be obtained given some knowledge of the distribution π. Such improved bounds have been obtained by Nakatsu [69], Sheinwald [78], Yeung [87], De Prisco and De Santis [18] (who consider, among else, dyadic distributions), Bose and Douïeb [11] (for general BSTs), and others. The paper of De Prisco and De Santis contains a mistake, which we correct in the full version of the paper.

Kleitman and Saks [51] considered the following problem: given a probability distribution π, what order for π results in the largest redundancy of the optimal alphabetic tree? Assuming $\pi_1 \ge \cdots \ge \pi_n$, they showed that the worse order is $x_n, x_1, x_{n-1}, x_2, \ldots$, and gave a formula for the cost of the optimal alphabetical tree for that order.

Computing the optimal length-restricted alphabetical tree has been studied extensively [8, 30, 40, 57, 75]. Dynamic alphabetical trees have been considered by Grinberg, Rajagopalan, Venkatesan and Wei [35]. See Nagaraj [68] for an extensive survey on these and other topics.

3.1.3 Binary search trees with comparison queries. As we have seen above, binary search trees involve a three-way comparison: "$x < c$, $x = c$, or $x > c$?". While modern programming languages usually support three-way comparisons, in the past a three-way comparison was implemented as two consecutive two-way comparisons: "$x = c$?" followed by "$x < c$?". This prompted Sheil [77] to suggest replacing the first comparison above by "$x = d$?", where d is the current most probable element. The resulting data structure is known as a *binary split tree*.

Huang and Wong [43] and Perl [71] (see also [38]) gave $O(n^5)$ dynamic programming algorithms that find the optimal binary split tree given a distribution on the elements, and this was improved to $O(n^4)$ by Chrobak et al. [14].

Huang and Wong [42] suggested relaxing the requirement that d (the element participating in the query "$x = d$?") be the current most probable element. The resulting data structure is known as a *generalized binary split tree*. They suggested a dynamic programming algorithm for finding the optimal generalized binary split tree, but Chrobak et al. [14] showed that their algorithm is invalid; no other efficient algorithm is known.

Spuler [81, 82] suggested uncoupling the two comparisons. His *two-way comparison tree* is a decision tree which uses comparison and equality queries. Anderson et al. [5] called this data structure a *binary comparison search tree* (BCST).

The notion of successful versus unsuccessful searches, which differentiates binary search trees and alphabetical trees (see above), also appears in the context of BCSTs. As an example, consider a domain with a single element c. If only successful searches are allowed, then the trivial BCST suffices. Otherwise, two nodes are needed to determine whether $x < c$, $x = c$, or $x > c$. The model considered in this paper only allows successful searches.

Spuler gave an $O(n^5)$ dynamic programming algorithm for finding the optimal BCST given a distribution on the successful searches, and Anderson et al. [5] improved this to $O(n^4)$. They also list several interesting properties of optimal BCSTs.

Chrobak et al. [14] generalized the algorithm of Anderson et al. to the case in which unsuccessful searches are allowed, and gave an $O(n \log n)$ algorithm based on weight-balancing which has redundnacy 3.

3.1.4 Searching in trees and posets. Ben-Asher, Farchi and Newman [9] consider the natural generalization of alphabetical trees to posets. Given a poset X, the the task is to find a secret element x

using queries of the form "$x \prec c$?". When X is a linear order, this is just an alphabetical tree.

Ben-Asher et al. concentrate on the case in which the Hasse diagram of X is a rooted tree, the root being the maximum element. In this case the queries can also be considered as edge queries: given an edge $e = (s, t)$, "Is x closer to s or to t?". If t is the node closer to the root, then this query is equivalent to the comparison query "$x \prec t$?".

Ben-Asher et al. give several applications of this model, to data transfer, software testing, and information retrieval. They are interested in minimizing the depth of the decision tree. Their main result is an $\tilde{O}n^4)$ algorithm which finds a decision tree of minimum depth, improved to $O(n^3)$ by Onak and Parys [70]. Lam and Yue [55] and Mozes, Onak and Weimann [67] give a linear time algorithm for the same problem.

Linial and Saks [58, 59] consider a different model for searching in posets: searching with partial information. In their model (adapted to our setting[3]) we are given a linearly ordered set X whose order is unknown, but is consistent with a known partial order on X. Given an element x, the task is to identify x using comparison queries. They are interested in minimizing the maximum number of queries.

As an example, suppose that A is an $n \times n$ matrix in which the rows and columns are increasing. How many queries are needed, in the worst case, to locate an element? They show that the answer is $2n - 1$, and relate this to the complexity of the merging procedure in merge sort.

Let i be the number of ideals (downward-closed sets) in the given partial order. Linial and Saks mention that $\log_2 i$ questions are needed in the worst case, and show that $O(\log_2 i)$ is achievable.

3.1.5 Sorting. Suppose we are given an array of length n having distinct elements. Sorting the array is the same as finding the relative order of the elements, which is a permutation $\pi \in S_n$. We can thus construe comparison-based sorting as the problem of finding a permutation $\pi \in S_n$ using queries of the form "$\pi(i) < \pi(j)$?". We call such queries *relative order queries*. We are usually interested in the worst case complexity of sorting algorithms, that is, the maximum number of relative order queries made by the algorithm.

Every comparison-based sorting algorithm must make at least $n \log_2 n - O(n)$ comparisons in the worst case. Merge sort comes very close, making $n \log_2 n + O(n)$ comparisons. In our terminology, the minimum depth of a decision tree for S_n using relative order queries is $n \log_2 n \pm \Theta(n)$.

Fredman [27] considered the problem of sorting with partial information. In this problem, we are given a set of permutations $\Gamma \subseteq S_n$, and the task is to identify a secret permutation $\pi \in \Gamma$ using relative order queries. Using the Gilbert–Moore algorithm, Fredman showed that this can be accomplished using at most $\log_2 |\Gamma| + 2n$ queries in the worst case.

Fredman applied his method to Berlekamp's $X + Y$ problem, which asks for sorting an array of the form $\{x_i + y_j : 1 \le i, j \le n\}$. Harper et al. [37] had improved on performance of merge sort (which uses $2n^2 \log_2 n + O(n^2)$ comparisons) by giving an algorithm which uses only $n^2 \log_2 n + O(n^2)$ comparisons. Fredman significantly improved on this by showing that $O(n^2)$ comparisons suffice, albeit the corresponding algorithm cannot be implemented efficiently. Lambert [56] gave an explicit algorithm using $O(n^2)$ comparisons, which also cannot be implemented efficiently.

When Γ consists of all completions of some partial order, Kahn and Saks [48] gave the upper bound $O(\log |\Gamma|)$ using the technique of poset balancing, but the corresponding algorithm cannot be implemented efficiently. Kahn and Kim [47] gave an algorithm with the same performance which can be implemented efficiently. Their algorithm relies on computing volumes of polytopes.

Moran and Yehudayoff [66] extended the results of Fredman to the distributional case. They showed that given a probability distribution μ on S_n, a secret permutation $\pi \sim \mu$ can be found using at most $H(\mu) + 2n$ relative order queries on average. In our terminology, they showed that the redundancy of relative order queries is at most $2n$. They actually proved a stronger bound: the number of queries required to identify π is at most $\log_2 \frac{1}{\mu(\pi)} + 2n$. This stronger bound implies Fredman's result.

Given a set of permutations $\Gamma \subseteq S_n$, endow S_n with the uniform permutation μ_Γ over Γ. A decision tree for Γ of depth $f(\log_2 \Gamma)$ is the same as a decision tree for μ_Γ of depth $f(H(\mu_\Gamma))$. In this way we can interpret all non-distributional results stated above as distributional results for uniform distributions.

3.1.6 Binary decision trees. Binary decision trees, or binary decision diagrams (BDDs), are used in statistics, machine learning, pattern recognition, data mining, and complexity theory. The setting is a set $X \subseteq \{0, 1\}^n$ and a function $f : X \to Y$ (for an arbitrary set Y). The task is to construct a decision tree which on input $x \in X$ computes $f(x)$ using only queries of the form "$x_i = 1$?" (*binary queries*).

In many settings, $Y = \{0, 1\}$, and such binary decision trees do not really fit our model. In other settings, $Y = X$ and f is the identity function. In this case these are decision trees in our sense which use binary queries.

We will not attempt to summarize the large literature on binary decision trees. We only mention the result of Hyafil and Rivest [45], who showed that it is NP-complete to compute the optimum cost of a decision tree using binary queries under the uniform distribution.

3.1.7 Algebraic decision trees. Algebraic decision trees [10] are commonplace in computational geometry. Given a set of real numbers $x_1, \ldots, x_n \in \mathbb{R}$ and a function f on $\mathbb{R}^n$, a (two-way) *linear decision tree* (LDT) is a decision tree for computing f using queries of the form "$\sum_i a_i x_i + a < 0$?" (*linear queries*); three-way variants also exist. *Algebraic decision trees* (ADTs) of order d are more general: they allow queries of the form "$P(x_1, \ldots, x_n) < 0$?" (*d'th order queries*), where P is an arbitrary polynomial of degree at most d. Interest has mainly focused on the worst-case cost of algebraic decision trees.

While algebraic decision trees, as just described, do not fit our model, the following variant does: given a finite set $S \subseteq \mathbb{R}^n$, the task is to find a secret element $x \in S$ using linear or d'th order queries.

Out of the voluminous literature on algebraic decision trees, we only mention the important paper of Ben-Or [10], which gives a lower bound of $\Omega(n \log n)$ on the depth of algebraic decision trees

[3] In their setting $X \subseteq \mathbb{R}$, and the task is to decide, given $x \in \mathbb{R}$, whether $x \in X$.

of fixed order which solve the element distinctness problem, using the Milnor–Thom theorem in real algebraic geometry.

A restricted type of linear decision trees occurs in the context of the 3SUM problem. In this problem, which is related to the $X + Y$ problem discussed above in the context of sorting, we are given three arrays X, Y, Z of size n, and our goal is to decide whether there exist $x \in X$, $y \in Y$, $z \in Z$ such that $x + y + z = 0$ (there are many other equivalent formulations). In the more general k-sum problem, we are given k arrays $X_1, \ldots, X_k$ of size n, and our goal is to decide whether there exist elements $x_i \in X_i$ such that $\sum_{i=1}^{k} x_i = 0$.

A classical algorithm solves 3SUM in time $O(n^2)$ by first sorting X and Y, and then, for each $z \in Z$, checking whether X and $-z - Y$ intersect using the merging procedure of merge sort. The classical algorithm can be construed as a three-way decision tree using queries of the form "$a_i < a_j$?" and "$a_i + a_j + a_k < 0, a_i + a_j + a_k = 0$, or $a_i + a_j + a_k > 0$?", where $a_1, \ldots, a_{3n}$ are the elements of X, Y, Z.

More generally, a (two-way) *s-linear decision tree* is a decision tree using questions of the form "$\sum_i c_i a_i < 0$?", where at most s of the c_i are non-zero; a three-way version is defined analogously. The classical algorithm uses a three-way 3-linear decision tree.

Erickson [23] proved a lower bound of $\Omega(n^{(k+1)/2})$ on the depth of a k-linear decision tree solving k-SUM when k is odd, and a lower bound of $\Omega(n^{k/2})$ when k is even. Weaker lower bounds for s-linear decision trees when $s > k$ were proved by Ailon and Chazelle [4].

The 3SUM conjecture (in one formulation) states that 3SUM cannot be solved in time $\Omega(n^{2-\epsilon})$ for any $\epsilon > 0$ (in the RAM machine model). Indeed, the stronger lower bound of $\Omega(n^2)$ had been conjectured. Similar lower bounds (matching the exponents stated above) exist for k-SUM.

Recently, in a breakthrough result, Grønlund and Pettie [36] gave a $o(n^2)$ algorithm for 3SUM (see Freund [28] for a simplification). Their algorithm is based on a 4-linear decision tree for 3SUM whose depth is $\tilde{O}(n^{3/2})$. More generally, for odd k they gave a $(2k-2)$-linear decision tree for k-SUM whose depth is $\tilde{O}(n^{k/2})$.

Meyer auf der Heide [63] gave a linear decision tree for k-SUM whose depth is $\tilde{O}(n^4)$, for any constant k. Recently, Cardinal et al. [13] improved this to $\tilde{O}(n^3)$ using a technique of Meiser [62], and Ezra and Sharir [25] improved the bound to $\tilde{O}(n^2)$. These authors solve a variant of the k-SUM problem in which we are given only *one* list $x_1, \ldots, x_n$, and the task is to decide whether k of its elements sum to zero. Their methods also solve a generalization, *k-linear degeneracy testing* (k-LDT), which asks whether there exist k indices $i_1 < \cdots < i_k$ such that $a_0 + a_1 x_{i_1} + \cdots + a_k x_{i_k} = 0$, for some constants $a_0, \ldots, a_k$.

3.2 Other Topics

We close the literature review by mentioning a few other related topics.

Combinatorial search theory. The "20 questions" game is the starting point of combinatorial search theory. Well-known examples include counterfeit coin problems [79] (also known as balance problems). See the survey by Katona [50], the monograph of Ahlswede and Wegener [3], and the recent volume [7] in memory of Ahlswede. Combinatorial search theory considers many different variants of the "20 questions" game, such as several unknown elements, non-adaptive queries, non-binary queries, and a non-truthful Alice; we expand below on the latter variant. Both average-case and worst-case complexity measures are of interest.

An important topic in combinatorial search theory is *combinatorial group testing*, in which we want to identify a set of at most d defective items out of n items using as few tests as possible. The original motivation was blood testing [20]. See the monograph by Du and Hwang [21]. Combinatorial group testing is related to the area of *combinatorial designs*, see for example Colbourn et al. [16].

Playing 20 questions with a liar. We have described a distributional version of the "20 questions" game in the introduction. The more usual version has Alice pick an object x from a known finite set X of size n. Bob is then tasked with discovering the secret object x using as few Yes/No questions as possible (in the worst case). If Bob is allowed to ask arbitrary questions, his optimal strategy reveals x using $\lceil \log_2 n \rceil$ questions at most.

In the usual version of the game, Alice is truthful. Rivest et al. [74] considered the case in which Alice is allowed to lie k times. They gave a strategy which asks at most $\log_2 n + k \log_2 \log_2 n + O(k \log k)$ questions in the worst case, all of them comparison queries. Moreover, they showed that $\log_2 n + k \log_2 \log_2 n + O(k \log k)$ questions are necessary, even if arbitrary questions are allowed.

Another line of work, which allows for a constant fraction r of lies, culminated in the work of Spencer and Winkler [80], who determined the threshold r for which Bob can win in several different scenarios. Aslam and Dhagat [6] and Dhagat et al. [19] analyze the same scenarios under various restricted sets of questions

Another line of work, which allows for a constant fraction r of lies, culminated in the work of Spencer and Winkler [80], who determined the threshold r for which Bob can win in the following three scenarios:

- Batch game: Alice knows Bob's strategy, and is allowed to lie in an r-fraction of answers.
- Adaptive game: Alice doesn't know Bob's strategy (which can depend on Alice's answers), and is allowed to lie in an r-fraction of answers.
- Prefix-bounded game: Alice doesn't know Bob's strategy, and is allowed to lie at most rm times in the first m questions, for every m.

Aslam and Dhagat [6] and Dhagat et al. [19] analyze these scenarios under various restricted sets of questions, including:

- Bit queries: "Is the ith bit of the binary representation of x equal to 1?".
- Comparison queries, which they also call cut queries.
- Tree queries: "Is $i2^j \le x < (i+1)2^j$?".

Asymmetric communication. One practical motivation for the (distributional) "20 questions" game is a communication scenario in which two entities, a client (Alice) and a server (Bob), communicate, and the uplink for the client to the server is much more costly than the downlink from the server to the client.

Adler and Maggs [2] suggest a formalization of this setting. In their model, the client holds a string x, the server has black-box access to a distribution π on the set of possible strings, and the server's goal is to discover x, assuming it is drawn from π. The

server is allowed to access π by determining, for any string t, the probability that a string drawn from π has t as a prefix. An algorithm is measured using four different parameters:

- The expected number of bits sent by the server.
- The expected number of bits sent by the client.
- The expected number of black box accesses.
- The expected number of rounds of communication.

Adler and Maggs give various algorithms exploring the various trade-offs. Watkinson et al. [86], in follow-up work, give more algorithms, among them one based on comparison queries and the Gilbert–Moore algorithm.

Weight balancing. The weight balancing algorithm of Walker and Gottlieb [85] and Rissanen [73] (mentioned above in the section on binary search trees) is an instance of the more general *splitting* heuristic of Garey and Graham [31], also known as *generalized binary search* (GBS).

Given an arbitrary set of queries, at any given moment the splitting heuristic chooses the query which minimizes $|\Pr[Yes]-\Pr[No]|$. This is a special case of the well-known ID3 algorithm [72], used in statistics, machine learning and data mining; the ID3 algorithm also handles k-way queries.

Kosaraju et al. [54] analyzed the performance of the splitting heuristic on general sets of questions. They showed that a slight modification of the heuristic gives an $O(\log n)$ approximation for the optimal cost using the given set of questions, where $n = |X|$ is the size of the domain.

A CORRECT VERSION OF *ON BINARY SEARCH TREES*

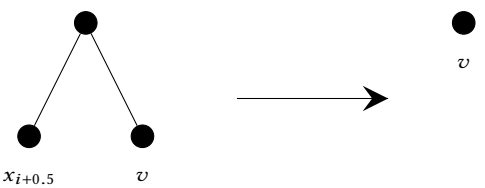
De Prisco and De Santis, in their paper *On binary search trees* [18], claim that $r^H(Q_\prec, \pi) \le 1 + p_{\max}$, where $p_{\max}$ is the maximum probability of an element in π. However, this is wrong even for the simple distribution $0, 1/3, 0, 1/3, 0, 1/3, 0$ (we require the algorithm to identify correctly even the zero probability elements), as a simple calculation reveals. Nevertheless, their ideas can be used to prove the following result:

Lemma A.1. *For any distribution π over X_n,*

$$r^H(Q_\prec, \pi) < 1 - \frac{\pi_1 + \pi_n}{2} + \frac{1}{2}\sum_{i=1}^{n-1} |\pi_{i+1} - \pi_i|.$$

Proof. Let $Y = x_{0.5} \prec x_1 \prec x_{1.5} \prec x_2 \prec \cdots \prec x_{n-0.5} \prec x_n \prec x_{n+0.5}$, and extend π to a distribution σ by giving all new elements zero probability. The Gilbert–Moore algorithm [34] produces a decision tree using $Q_\prec$ whose cost is less than $H(\sigma) + 2 = H(\pi) + 2$.

The decision tree contains $n+1$ redundant leaves, corresponding to the newly added elements. We get rid of them one by one:



Getting rid of $x_{i+0.5}$ in this way bumps up at least one of x_i, x_{i+1} by one level; when $i = 0$ or $i = n$, only one of these options is available. In total, we bump up leaves whose total probability is at least

$$\pi_1+\pi_n+\sum_{i=1}^{n-1}\min(\pi_i,\pi_{i+1}) = \pi_1+\pi_n+\sum_{i=1}^{n-1}\frac{\pi_i+\pi_{i+1}-|\pi_i-\pi_{i+1}|}{2}$$
$$= 1 + \frac{\pi_1+\pi_n}{2} - \frac{1}{2}\sum_{i=1}^{n-1}|\pi_i-\pi_{i+1}|.$$

The cost of the pruned tree is thus as stated. Replacing each question $\prec x_{i+0.5}$ by the equivalent question $\prec x_{i+1}$, we get a legal decision tree for the original distribution π using $Q_\prec$. □

REFERENCES

[1] Julia Abrahams. 1997. Code and parse trees for lossless source encoding. In *Compression and Complexity of Sequences*. 145–171.

[2] Micah Adler and Bruce M. Maggs. 2001. Protocols for Asymmetric Communication Channels. *J. Comput. System Sci.* 63, 4 (2001), 573–596.

[3] Rudolf Ahlswede and Ingo Wegener. 1987. *Search problems.* John Wiley & Sons, Inc., New York.

[4] Nir Ailon and Bernard Chazelle. 2005. Lower bounds for linear degeneracy testing. *J. ACM* 52, 2 (2005), 151–171.

[5] Richard Anderson, Sampath Kannan, Howard Karloff, and Richard E. Ladner. 2002. Thresholds and optimal binary comparison search trees. *Journal of Algorithms* 44 (2002), 338–358.

[6] Javad A. Aslam and Aditi Dhagat. 1991. Searching in the presence of linearly bounded errors. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing (STOC '91)*. 486–493.

[7] Harout Aydinian, Ferdinando Cicalese, and Christian Deppe (Eds.). 2013. *Information Theory, Combinatorics, and Search Theory.* Springer-Verlag Berlin Heidelberg.

[8] Michael B. Baer. 2007. Twenty (or so) Questions: D-ary Length-Bounded Prefix Coding. In *IEEE International Symposium on Information Theory (ISIT 2007)*. 896–900.

[9] Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. 1999. Optimal search in trees. *SIAM J. Comput.* 28, 6 (1999), 2090–2102.

[10] Michael Ben-Or. 1983. Lower bounds for algebraic computation trees. In *Proceedings of the 15th ACM Symposium on Theory of Computing*. 80–86.

[11] Prosenjit Bose and Karim Douïeb. 2009. Efficient Construction of Near-Optimal Binary and Multiway Search Trees. In *Algorithms and Data Structures (WADS 2009)*. 230–241.

[12] Renato M. Capocelli, Raffaele Giancarlo, and Indeer Jeet Taneja. 1986. Bounds on the Redundancy of Huffman Codes. *IEEE Transactions on Information Theory* IT-32, 6 (1986), 854–857.

[13] Jean Cardinal, John Iacono, and Aurélien Ooms. 2016. Solving k-SUM using few linear queries. In *24th Annual European Symposium on Algorithms (ESA 2016)*. LIPIcs, 25:1–25:17.

[14] Marek Chrobak, Mordecai Golin, J. Ian Munro, and Neal E. Young. 2015. Optimal Search Trees with 2-Way Comparisons. In *Algorithms and Computation (ISAAC 2015)*. 71–82.

[15] David Cohn, Les Atlas, and Richard Ladner. 1994. Improving Generalization with Active Learning. *Machine Learning* 15 (1994), 201–221.

[16] C. J. Colbourn, J. H. Dinitz, and D. R. Stinson. 1993. Applications of combinatorial designs to communications, cryptography, and networking. In *Surveys in Combinatorics, 1993*, K. Walker (Ed.). London Mathematical Society Lecture Note Series, Vol. 187. Cambridge University Press.

[17] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of information theory (2. ed.).* Wiley.

[18] Roberto De Prisco and Alfredo De Santis. 1993. On Binary Search Trees. *Inform. Process. Lett.* 45, 5 (1993), 249–253.

[19] Aditi Dhagat, Peter Gács, and Peter Winkler. 1992. On Playing "Twenty Questions" with a Liar. In *Proceedings of 3rd Symposium on Discrete Algorithms (SODA'92)*. 16–22.

[20] Robert Dorfman. 1943. The Detection of Defective Members of Large Populations. *The Annals of Mathematical Statistics* 14, 4 (1943), 436–440.

[21] Ding-Zhu Du and Frank K. Hwang. 1999. *Combinatorial Group Testing and Its Applications* (2nd ed.). Series on Applied Mathematics, Vol. 12. World Scientific.

[22] Dwight Duffus, Bill Sands, and Peter Winkler. 1990. Maximal chains and antichains in Boolean lattices. *SIAM Journal on Discrete Mathematics* 3, 2 (1990), 197–205.

[23] Jeff Erickson. 1999. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science* 8 (1999).

[24] William Evans and David Kirkpatrick. 2004. Restructuring ordered binary trees. *Journal of Algorithms* 50, 2 (2004), 168–193.

[25] Ester Ezra and Micha Sharir. 2016. The Decision Tree Complexity for k-SUM is at most Nearly Quadratic. Manuscript. (2016).

[26] Newton Faller. 1973. An adaptive system for data compression. In *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*. 593–597.

[27] Michael L. Fredman. 1976. How good is the information theory bound in sorting? *Theoretical Computer Science* 1, 4 (1976), 355–361.

[28] Ari Freund. 2015. Improved Subquadratic 3SUM. *Algorithmica* (2015), 1–19.

[29] Robert G. Gallager. 1978. Variations on a Theme by Huffman. *IEEE Transactions on Information Theory* IT-24, 6 (1978), 668–674.

[30] Michael R. Garey. 1974. Optimal Binary Search Trees with Restricted Maximal Depth. *SIAM J. Comput.* 3, 2 (1974), 101–110.

[31] Michael R. Garey and Ronald L. Graham. 1974. Performance bounds on the splitting algorithm for binary testing. *Acta Informatica* 3 (1974), 347–355.

[32] Adriano M. Garsia and Michelle L. Wachs. 1977. A new algorithm for minimum cost binary trees. *SIAM J. Comput.* 6, 4 (1977), 622–642.

[33] Edgar N. Gilbert. 1971. Codes based on inaccurate source probabilities. *IEEE Transactions on Information Theory* 17, 3 (1971), 304–314.

[34] E. N. Gilbert and E. F. Moore. 1959. Variable-Length Binary Encodings. *Bell System Technical Journal* 38 (1959), 933–967.

[35] Dennis Grinberg, Sivaramakrishnan Rajagopalan, Ramarathnam Venkatesan, and Vicor K. Wei. 1995. Splay Trees for Data Compression. In *Proceedings of the sixth annual ACM–SIAM symposium on Discrete algorithms (SODA'95)*. 522–530.

[36] Allan Grønlund and Seth Pettie. 2014. Threesomes, Degenerates, and Love Triangles. In *55th Annual Symposium on Foundations of Computer Science (FOCS'14)*. 621–630.

[37] L. H. Harper, T. H. Payne, J. E. Savage, and E. Straus. 1975. Sorting X+Y. *Commun. ACM* 18, 6 (1975), 347–349.

[38] James H. Hester, Daniel S. Hirschberg, Shou-Hsuan Stephen Huang, and C. K. Wong. 1986. Faster Construction of Optimal Binary Split Trees. *Journal of Algorithms* 7, 3 (1986), 412–424.

[39] Yasuichi Horibe. 1977. An improved bound for weight-balanced tree. *Information and Control* 34, 2 (1977), 148–151.

[40] T. C. Hu and K. C. Tan. 1972. Path Length of Binary Search Trees. *SIAM J. Appl. Math.* 22, 2 (1972), 225–234.

[41] T. C. Hu and Alan Curtiss Tucker. 1971. Optimal computer search trees and variable length alphabetic codes. *Journal of Applied Mathematics* 21 (1971), 514–532.

[42] Shou-Hsuan Stephen Huang and C. K. Wong. 1984. Generalized binary split trees. *Acta Informatica* 21, 1 (1984), 113–123.

[43] Shou-Hsuan Stephen Huang and C. K. Wong. 1984. Optimal binary split trees. *Journal of Algorithms* 5, 1 (1984), 69–79.

[44] David A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. In *Proceedings of the I.R.E.* 1098–1103.

[45] Laurent Hyafil and Ronald L. Rivest. 1976. Constructing optimal binary decision trees is NP-complete. *Inform. Process. Lett.* 5, 1 (1976), 15–17.

[46] Ottar Johnsen. 1980. On the Redundancy of Binary Huffman Codes. *IEEE Transactions on Information Theory* IT-26, 2 (1980), 220–222.

[47] Jeff Kahn and Jeong Han Kim. 1995. Entropy and Sorting. *J. Comput. System Sci.* 51, 3 (1995), 390–399.

[48] Jeff Kahn and Michael Saks. 1984. Balancing poset extensions. *Order* 1, 2 (1984), 113–126.

[49] Richard M. Karp. 1961. Minimum-redundancy coding for the discrete noiseless channel. *I.R.E. Transactions on Information Theory* (1961), 27–38.

[50] Gyula O. H. Katona. 1973. Combinatorial Search Problems. In *A Survery of Combinatorial Theory*, J. N. Srivastava et al. (Ed.). North-Holland Publishing Company.

[51] Daniel J. Kleitman and Michael E. Saks. 1981. Set orderings requiring costliest alphabetic binary trees. *SIAM Journal on Algebraic Discrete Mathematics* 2, 2 (1981), 142–146.

[52] Donald E. Knuth. 1971. Optimum binary search trees. *Acta Informatica* 1, 1 (1971), 14–25.

[53] Donald E. Knuth. 1985. Dynamic Huffman coding. *Journal of Algorithms* 6 (1985), 163–180.

[54] S. Rao Kosaraju, Teresa M. Przytycka, and Ryan Borgstrom. 1999. On an Optimal Split Tree Problem. In *Algorithms and Data Structures: 6th International Workshop, WADS'99 Vancouver, Canada, August 11–14, 1999 Proceedings*, Frank Dehne, Jörg-Rüdiger Sack, Arvind Gupta, and Roberto Tamassia (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 157–168. DOI: http://dx.doi.org/10.1007/3-540-48447-7_17

[55] T. W. Lam and F. L. Yue. 2001. Optimal edge ranking of trees in linear time. *Algorithmica* 30, 1 (2001), 12–33.

[56] Jean-Luc Lambert. 1992. Sorting the sums $(x_i + y_j)$ in $O(n^2)$ comparisons. *Theoretical Computer Science* 103, 1 (1992), 137–141.

[57] Lawrence L. Larmore. 1987. Height-restricted optimal binary trees. *SIAM J. Comput.* 16 (1987), 1115–1123.

[58] Nati Linial and Michael Saks. 1985. Every poset has a central element. *Journal of Combinatorial Theory* 40 (1985), 195–210.

[59] Nati Linial and Michael Saks. 1985. Searching order structures. *Journal of Algorithms* 6 (1985), 86–103.

[60] Zbigniew Lonc and Ivan Rival. 1987. Chains, Antichains, and Fibres. *Journal of Combinatorial Theory, Series A* 44 (1987), 207–228.

[61] Dietrich Manstetten. 1992. Tight Bounds on the Redundancy of Huffman Codes. *IEEE Transactions on Information Theory* IT-38, 1 (1992), 144–151.

[62] S. Meiser. 1993. Point location in arrangements of hyperplanes. *Information and Computation* 106, 2 (1993), 286–303.

[63] Friedhelm Meyer auf der Heide. 1984. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM* 31 (1984), 668–676.

[64] Soheil Mohajer, Payam Pakzad, and Ali Kakhbod. 2006. Tight Bounds on the Redundancy of Huffman Codes. In *Information Theory Workshop (ITW '06)*. 131–135.

[65] Bruce L. Montgomery and Julia Abrahams. 1987. On the Redundancy of Optimal Binary Prefix-Condition Codes for Finite and Infinite Sources. *IEEE Transactions on Information Theory* IT-33, 1 (1987), 156–160.

[66] Shay Moran and Amir Yehudayoff. 2016. A note on average-case sorting. *Order* 33, 1 (2016), 23–28.

[67] Shay Mozes, Krzysztof Onak, and Oren Weimann. 2008. Finding an optimal tree searching strategy in linear time. In *Proceedings of 19th Symposium on Discrete Algorithms (SODA'08)*. 1096–1105.

[68] S. V. Nagaraj. 1997. Optimal binary search trees. *Theoretical Computer Science* 188, 1–2 (1997), 1–44.

[69] Narao Nakatsu. 1991. Bounds on the Redundancy of Binary Alphabetical Codes. *IEEE Transactions on Information Theory* IT-37, 4 (1991), 1225–1229.

[70] Krzysztof Onak and Paweł Parys. 2006. Generalization of Binary Search: Searching in Trees and Forest-Like Partial Orders. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. 379–388. DOI: http://dx.doi.org/10.1109/FOCS.2006.32

[71] Yehoshua Perl. 1984. Optimum split trees. *Journal of Algorithms* 5, 3 (1984), 367–374.

[72] J. Ross Quinlan. 1986. Induction of Decision Trees. *Machine Learning* 1 (1986), 81–106.

[73] Jorma Rissanen. 1973. Bounds for weight balanced trees. *IBM Journal of Research and Development* 17 (1973), 101–105.

[74] Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. 1980. Coping with errors in binary search procedures. *J. Comput. System Sci.* 20 (1980), 396–404.

[75] Baruch Schieber. 1998. Computing a Minimum Weight k-Link Path in Graphs with the Concave Monge Property. *Journal of Algorithms* 29 (1998), 204–222.

[76] Claude Elwood Shannon. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal* 27 (1948), 379–423.

[77] B. A. Sheil. 1978. Median split trees: a fast lookup technique for frequently occuring keys. *Commun. ACM* 21, 11 (1978), 947–958.

[78] Dafna Sheinwald. 1992. On binary alphabetical codes. In *Data Compression Conference (DCC'92)*. 112–121.

[79] Cedric A. B. Smith. 1947. The Counterfeit Coin Problem. *The Mathematical Gazette* 31, 293 (1947), 31–39.

[80] Joel Spencer and Peter Winkler. 1992. Three thresholds for a liar. *Combinatorics, Probability and Computing* 1, 1 (1992), 81–93.

[81] David Spuler. 1994. Optimal search trees using two-way key comparisons. *Acta Informatica* 31 (1994), 729–740.

[82] David A. Spuler. 1994. *Optimal Binary Trees With Two-Way Key Comparisons*. Ph.D. Dissertation. James Cook University.

[83] Jan van Leeuwen. 1987. On the construction of Huffman trees. In *Third International Colloquium on Automata, Languages and Programming (ICALP '87)*. 382–410.

[84] Jeffrey Scott Vitter. 1987. Design and analysis of dynamic Huffman codes. *J. ACM* 34, 4 (1987), 825–845.

[85] W. A. Walker and C. C. Gottlieb. 1972. *A top-down algorithm for constructing nearly optimal lexicographical trees*. Academic Press, 303–323.

[86] John Watkinson, Micah Adler, and Faith E. Fich. 2001. New Protocols for Asymmetric Communication Channels. In *8th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 337–350.

[87] Raymond W. Yeung. 1991. Alphabetic codes revisited. *IEEE Transactions on Information Theory* IT-37, 3 (1991), 564–572.