Scenario-Based Robust Optimization of Tree Structures

Spyros Angelopoulos[2, 3], **Christoph Dürr**[1, 2], **Alex Elenter**[1], **Georgii Melidi**[1]

[1]Sorbonne University, LIP6, Paris, France
[2]CNRS
[3]International Laboratory on Learning Systems, Montreal, Canada
{spyros.angelopoulos, christoph.durr, alex.elenter, georgii.melidi}@lip6.fr

Abstract

We initiate the study of tree structures in the context of scenario-based robust optimization. Specifically, we study Binary Search Trees (BSTs) and Huffman coding, two fundamental techniques for efficiently managing and encoding data based on a known set of frequencies of keys. Given a number of distinct scenarios, each defined by a frequency distribution over the keys, our objective is to compute a single tree of best-possible performance, relative to any scenario.

We consider, as performance metrics, the competitive ratio, which compares multiplicatively the cost of the solution to the tree of least cost among all scenarios, as well as the regret, which induces a similar, but additive comparison. For BSTs, we show that the problem is NP-hard across both metrics. We also obtain an optimal competitive ratio that is logarithmic in the number of scenarios. For Huffman Trees, we likewise prove NP-hardness, and we present an algorithm with logarithmic regret, which we prove to be near-optimal by showing a corresponding lower bound. Last, we give a polynomial-time algorithm for computing Pareto-optimal BSTs with respect to their regret, assuming scenarios defined by uniform distributions over the keys. This setting captures, in particular, the first study of fairness in the context of data structures. We provide an experimental evaluation of all algorithms. To this end, we also provide mixed integer linear program formulation for computing optimal trees.

Code — https://gitlab.lip6.fr/gmelidi/robust-tree
Extended version — https://arxiv.org/pdf/2408.11422

1 Introduction

Suppose that we would like to encode and transmit a text in a given language efficiently, i.e., using the least number of bits on expectation. If the alphabet's frequency is known ahead of time, i.e., if the language is pre-determined, this can be done efficiently using the well-known technique of *Huffman coding* (Huffman 1952). But what if we do not know in advance the intended language, but instead it is only known that it can be either English, Italian, or Finnish? In this case, one would like to design a *single* code that performs efficiently across all three such scenarios, and in particular against a worst-case, adversarially chosen language.

For a second example, suppose that we are given n keys to be stored in a *Binary Search Tree* (BST), and that we would like to minimize the expected number of comparisons, when performing a search operation for a key. Once again, if the key frequencies are known ahead of time, finding an *optimal* BST is a fundamental problem in Computer Science, going back to the seminal work of Knuth (Knuth 1971) who gave a quadratic-time algorithm. Suppose, however, that instead of a single frequency vector, we are provided with k different vectors, each defining one possible scenario. How would we construct a *single* BST that performs well across all scenarios, hence also against one chosen adversarially?

Motivated by the above situations, in this work we introduce the study of *robust* structures, in the presence of k possible *scenarios*. Each scenario is described by means of a frequency vector over the keys, and we seek a single tree that is robust with respect to *any* scenario (and thus with respect to the worst-case, adversarial scenario). This approach falls into what is known as *scenario-based robust optimization*; see (Ben-Tal, El Ghaoui, and Nemirovski 2009) for a survey of this area. While the scenario-based framework has been studied in the context of AI-related optimization problems in areas such as planning (McCarthy, Vayanos, and Tambe 2017), scheduling (Shabtay and Gilenson 2023) and network optimization (Kasperski and Zieliński 2015; Kasperski, Kurpisz, and Zieliński 2015), to our knowledge it has not been applied to data structures or data coding.

We consider two main measures for evaluating the quality of our solutions. The first measure applies to BSTs, and is the worst-case ratio, among all scenarios s, between the cost of our solution (tree) under scenario s and the cost of the optimal tree under s; using the canonical term from online computation, we refer to this measure as the *competitive ratio*. For Huffman trees, we consider even stronger guarantees, by studying the worst-case *difference*, among all scenarios s between the cost of our solution under s and the cost of the optimal tree for s, namely the *regret* of the solution. We refer to Sections 2.1 and 3.1 for the formal definitions. Competitive analysis and regret minimization are both well-studied approaches in optimization under uncertainty, that establish strict, worst-case guarantees under adversarial situations (Borodin and El-Yaniv 2005), (Blum and Mansour 2007). Competitive analysis is the predominant analysis technique for tree-based data structures, see

the seminal work (Sleator and Tarjan 1985) on Splay Trees. Regret minimization, on the other hand, may provide more stringent guarantees (c.f. Corollary 3.5), and notions related to regret have been applied, for instance, to the evaluation of query efficiency in databases (Xie, Wong, and Lall 2020), (Nanongkai et al. 2010).

1.1 Contribution

We begin with the study of robust BSTs in Section 2. We first show that minimizing either the competitive ratio or the regret is NP-hard, even if there are only two scenarios, i.e., $k = 2$ (Theorem 2.3). We also give an algorithm that constructs a BST of competitive ratio at most $\lceil \log_2(k+1) \rceil$ (Theorem 2.4), and we show that this bound is optimal, in that there exists a collection of k scenarios under which no BST can perform better (Theorem 2.6).

In Section 3, we study robust Huffman trees (HTs). We first show that the problem of minimizing the competitive ratio or the regret is NP-hard, again even if $k = 2$ (Theorem 3.1). We also give an algorithm for constructing a Huffman tree that has regret that is provably at most $\lceil \log_2 k \rceil$. We show that this is essentially optimal, by proving a near-matching lower bound equal to $\lfloor \log_2 k \rfloor$ (Theorem 3.4).

In Section 4, we study the problem of minimizing BST regret with respect to k scenarios. This problem can be formulated as a k-objective optimization problem, hence we seek trees in the *Pareto-frontier* of the k-objectives. For concreteness, we focus on scenarios induced by uniform distributions over subsets of keys: here, we give a polynomial-time algorithm for finding Pareto-optimal solutions. This formulation provides the first framework for quantifying *fairness* in the context of data structures. More precisely, we can think of each distinct scenario as the profile of a different *user*, and the BST as the single *resource* that is shared by all k users. We thus seek a solution that distributes the cost as equitably as possible among the k competing users.

In Section 5, we provide an experimental evaluation of all our algorithms over real data. To evaluate and compare our algorithms from Sections 2 and 3, we also provide mixed integer linear program formulations for all objectives, which allows us to compute the optimal trees for small instances.

In terms of techniques, despite the seeming similarity of the settings, we show that robust BSTs and HTs are quite different problems; this is due to the fact that in the former the keys are stored in all nodes, whereas in the latter they are stored in leaves. This is reflected both in the different NP-hardness proofs and in the different algorithmic approaches. Namely, for BSTs, NP-hardness is established using a non-trivial reduction from the PARTITION problem (Theorem 2.3), whereas for HTs we use a non-trivial reduction from the EQUAL-CARDINALITY PARTITION problem (Theorem 3.1). From the algorithmic standpoint, for BSTs we follow a recursive approach for constructing the tree, whereas for HTs we use an approach that allows us to "aggregate" the optimal HT for each scenario into a single HT. Last, in terms of finding the Pareto-frontier for regret minimization in BSTs, we use a dynamic programming approach that allows for an efficient implementation. We refer to (Angelopoulos et al. 2024) for the full paper version.

1.2 Related Work

The BST is a fundamental data structure that has been studied extensively since the 1960s. See (Windley 1960; Booth and Colin 1960; Hibbard 1962) for some classic references and (Nagaraj 1997) for a survey. Given a frequency vector of accesses to n different keys, finding a BST of optimal average access cost was originally solved in (Knuth 1971). Likewise, Huffman coding (Huffman 1952) is a fundamental technique for lossless data compression, which combines optimality of performance and simplicity of implementation. Given a frequency vector over an n-sized alphabet, a Huffman tree can be implemented in time $O(n \log n)$ using a priority queue. We refer to (Moffat 2019) for a survey.

A recent, and very active related direction in Machine Learning seeks to augment data structures with some *prediction* concerning the future access requests. Examples of learning-augmented data structures that have been studied include skip lists (Fu, Seo, and Zhou 2024; Zeynali, Kamali, and Hajiesmaili 2024), BSTs (Lin, Luo, and Woodruff 2022; Cao et al. 2022), B-trees (Kraska et al. 2018) and rank/select dictionaries (Boffa, Ferragina, and Vinciguerra 2022); see also (Ferragina and Vinciguerra 2020) for a survey on the applications of ML in data structures. These works leverage a learned prediction about access patterns, and seek structures whose performance degrades smoothly as a function of the prediction error. The settings we study in this work can thus be interpreted, equivalently, as having access to k different predictions, and seeking structures that perform efficiently even if the worst-case prediction materializes.

It is known that if the access frequencies are chosen uniformly at random, then with high probability the optimal BST is very close to a complete binary search tree, and its expected height is logarithmic in the number of keys n. In the other extreme, there exist adversarial frequencies for which the optimal BST has height that is as large as $\Omega(n)$. Several works have studied the regime between these extremes, via small, random perturbations to the frequency vector, e.g., (Manthey and Reischuk 2007). Bicriteria optimization problems over BSTs were studied in (Mankowski and Moshkov 2020), with the two objectives being the maximum and the average weighted depth, respectively.

Scenario-based robust optimization has been extensively applied to the study of *scheduling* problems under uncertainty. Examples include the minimization of completion time (Mastrolilli, Mutsanas, and Svensson 2013), flow-shop scheduling (Kasperski, Kurpisz, and Zielinski 2012) and just-in-time scheduling (Gilenson and Shabtay 2021). We refer to (Shabtay and Gilenson 2023) for a recent survey of many results related to scenario-based scheduling.

We conclude with a discussion of *fairness*, which is becoming an increasingly demanded requirement in algorithm design and analysis. Algorithmic fairness, defined as an equitable treatment of users that compete for a common resource, has been studied in some classic settings, including optimal stopping problems (Arsenis and Kleinberg 2022), (Buchbinder, Jain, and Singh 2009), and resource allocation problems such as knapsack (Lechowicz et al. 2024; Patel, Khan, and Louis 2021). However, to our knowledge, no previous work has addressed the issue of fairness in the

context of data structures, although the problem is intrinsically well-motivated: e.g., we seek a data structure (such as a BST) that guarantees an equitable search time across the several competing users that may have access to it. The concept of Pareto-dominance as a criterion for fairness has attracted attention in recent works in ML, e.g., (Martinez et al. 2021), (Martinez, Bertran, and Sapiro 2020). In our work, we do rely on learning oracles, and we seek Pareto-optimal solutions that capture group fairness based on regret.

2 Robust Binary Search Trees

2.1 Background and Measures

In its standard form, a BST stores n keys from a given ordered set; without loss of generality, we may assume that the set of keys is the set $\{1, \ldots, n\}$. The keys are stored in the nodes, and satisfy the *ordering* property: the key of any node is larger than all keys in its left sub-tree, and smaller than all keys in its right sub-tree. For a given key, the corresponding node is accessed in a recursive manner, starting from the *root*, and descending in the tree guided by key comparisons.

A BST can be conveniently represented, equivalently, by its *level vector*, denoted by L, in the sense that every key i has level L_i in the tree. Here we use the convention that the root has level 1. Formally, we have:

Definition 2.1. A vector $L \in \{1, \ldots, n\}^n$ is a *level vector* of a BST if and only if for every $1 \leqslant i < j \leqslant n$ with $L_i = L_j$, there is a key $i < r < j$ such that $L_r < L_i$.

The definition formulates the fact that if there are two keys at the same level of the BST, then there must exist a key between them of lower level (i.e., higher in the tree). This definition allows us to express the *average cost* of a BST represented by a level vector L, relative to a frequency vector F, as the inner product

$$\text{cost}(L, F) = \sum_{i=1}^{n} L_i \cdot F_i. \tag{1}$$

Given a frequency vector F, a level vector L minimizing $\text{cost}(L, F)$ can be computed, using dynamic programming, in time $O(n^2)$ (Knuth 1971).

Next, we define formally the *robust BST* problem. Its input consists of k frequency vectors $F^1, \ldots, F^k$, called *scenarios*. There are three possible performance metrics one could apply in order to evaluate the performance, which give rise to three possible minimization problems on the BST level vector L that must be found:

- *Worst-case cost*: here, the objective is to minimize $\max_s \text{cost}(L, F^s)$, i.e., we seek the tree of smallest cost under the worst-case scenario.
- *Competitive ratio*: here, we aim to minimize

$$\max_s \frac{\text{cost}(L, F^s)}{\min_{L^*} \text{cost}(L^*, F^s)},$$

i.e., we aim to minimize the worst-case ratio between the cost of our tree and the optimal tree for each scenario.

- *Regret*: here, the objective is to minimize the quantity

$$\max_s \{\text{cost}(L, F^s) - \min_{L^*} \text{cost}(L^*, F^s)\},$$

i.e., we want to achieve the smallest *difference* between the cost of our tree and the optimal tree for each scenario.

Note that if for each scenario the respective optimal trees have the same cost, then the three metrics are equivalent. However, in general, they may be incomparable, as shown in the following example.

Example 2.2. Let $F^1 = (0, 1/4, 3/4)$ and $F^2 = (4/9, 2/9, 1/3)$ denote two scenarios for three keys $\{a, b, c\}$. The various metrics for each possible BST are as depicted.

					
F^1	11/4	9/4	7/4	3/2	5/4
F^2	17/9	16/9	16/9	17/9	19/9
cost	11/4	9/4	16/9	17/9	19/9
c. ratio	11/5	9/5	7/5	6/5	19/16
regret	3/2	1	1/2	1/4	1/3

The first two rows of the table show the cost of each of the five possible BSTs on three keys, for the two scenarios F^1 and F^2; here the optimal costs for each scenario are highlighted in gray. The remaining three columns show the performance of each tree with respect to the three metrics, with the best performance highlighted in yellow.

Our hardness results, for both BSTs and HTs, apply to all three metrics. However, for the purposes of analysis, we focus on the competitive ratio and the regret. As discussed in Section 1, the competitive ratio is the canonical performance notion in the analysis of BSTs, and allows us to capture worst-case performance under uncertainty that can be efficiently approximated, both from the point of view of upper bounds (positive results) and lower bounds (impossibility results). In other words, the competitive ratio reflects the *price* of not knowing the actual scenario in advice, similar to the competitive analysis of online algorithms (Borodin and El-Yaniv 2005). On the other hand, regret-minimization can provide even stronger guarantees for HTs, but can also help model issues related to fairness in BSTs, as we discuss in detail in Section 4.

2.2 Results

We first show that finding an optimal robust BST is NP-hard, even if $k = 2$. We will prove the result for the cost-minimization version; however the proof carries over to the other metrics.

To prove NP-hardness, we first need to formulate the decision variant of the problem: Given two frequency vectors, F^1 and F^2, and a threshold V, the objective is to decide whether there exists a BST L of cost at most V, in either scenario, i.e., $\text{cost}(L, F^1) \leq V$ and $\text{cost}(L, F^2) \leq V$.

Theorem 2.3. *The robust BST problem is NP-hard, even if $k = 2$. This holds for all three metrics, i.e., for minimizing the cost, or the competitive ratio, or the regret.*

Proof. The proof is based on a reduction from the PARTITION problem (Garey and Johnson 1979). An instance of this problem consists of a list $a_1, ..., a_m$ of non-negative integers, and the goal is to decide whether there exists a binary vector $b \in \{0, 1\}^m$ with $\sum_i b_i a_i = \sum_i (1 - b_i) a_i$.

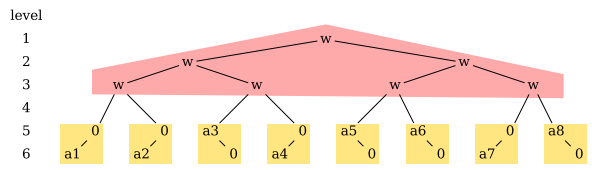

Figure 1: The BST corresponding to the binary vector $b = \{11010010\}$ in the NP-hardness proof construction. Nodes are labeled with the frequency of their keys in F^1.

Without loss of generality we can assume that m is of the form $m = 2^\ell$ for some integer ℓ, as we can always pad the instance with zeros.

From the given instance of the partition problem, we define an instance of the robust BST problem which consists of $n = 3m - 1$ keys and two frequency vectors $F^1 = (a_1, 0, w, a_2, 0, w, a_3, 0, \ldots, w, a_m, 0)$ and $F^2 = (0, a_1, w, 0, a_2, w, 0, a_3, \ldots, w, 0, a_m)$, where w is any constant strictly larger than $(\ell + 3/2)\sum_i a_i$. Moreover, the instance specifies the threshold $V = W + (\sum a_i)/2$, where W is defined as

$$W := w \sum_{j=1}^{\ell} j 2^{j-1} + (\ell + 1) \sum_{i=1}^{n} a_i.$$

We claim that the level vector L which optimizes $\max\{\text{cost}(L, F^1), \text{cost}(L, F^2)\}$ has the following structure. The first ℓ levels form a complete binary tree over all keys $3i$ for $i = 1, 2, \ldots, m$. These are exactly the keys with frequency w in both F^1 and F^2. (Intuitively, since w is large, these keys should be placed at the smallest levels.) Furthermore, for each $i = 1, 2, \ldots, m$, the keys $3i-2$ and $3i-1$ are placed at levels $\ell+1$ and $\ell+2$, or at levels $\ell+2$ and $\ell+1$, respectively. Such a tree has a cost at least W in each scenario and at most $W + \sum a_i$. The claim that L has this structure follows from the fact that if some key with frequency w were to be placed below level ℓ, then the tree would incur a cost of at least $w(1 + \sum_{j=1}^{\ell} j 2^{j-1})$, which is strictly greater than V. See Figure 1 for an illustration.

As a result, all solutions to the robust BST problem can be described by a binary vector $b \in \{0, 1\}^m$, such that for every $i = 1, 2, \ldots, m$ the key $3i - 2$ has level $\ell + 1 + b_i$ while key $3i - 1$ has level $\ell + 2 - b_i$. We denote by L^b the level vectors of these trees in order to express the costs as $\text{cost}(L^b, F^1) = W + \sum_i b_i a_i$ and $\text{cost}(L^b, F^2) = W + \sum_i (1 - b_i) a_i$.

Observe that if both costs of L^b are at most V, then they are equal to this value and b is a solution to the partition problem. This implies that deciding if there exists a BST of cost at most V is NP-hard. □

Next, we present an algorithm that achieves the optimal competitive ratio. Algorithm 1 first computes optimal trees with level vector L^s for each scenario $s = 1, \ldots, k$, then calls a recursive procedure A with initial values $i = 1$, $j = n$ and $\ell = 1$. Procedure A solves a subproblem defined by i, j, ℓ. Namely, it constructs a BST on the keys $[i, j]$ and stores its root at level ℓ. To this end, it first identifies

Algorithm 1: R-BST

Input: k scenarios described by frequency vectors $F^1, \ldots, F^k$.
Output: A robust BST represented as a level vector L.

1: For each scenario $s \in \{1, \ldots, k\}$ compute the optimal tree L^s
2: $A(i = 1, j = n, \ell = 1)$

Algorithm 2: Recursive procedure A

1: **procedure** A(i, j, ℓ)
2: **if** $i > j$ **then**
3: **return** ▷ Empty interval $[i, j]$ ends recursion
4: **else**
5: Let v be the minimum level L^s_r over $i \leqslant r \leqslant j$ and $1 \leqslant s \leqslant k$.
6: Let S be the ordered set of keys $i \leqslant r \leqslant j$ such that there exists a scenario s with $L^s_r = v$.
7: Let $m \in S$ be s.t. both $|S \cap [i, m-1]|$ and $|S \cap [m+1, j]|$ are at most $\lceil (|S| - 1)/2 \rceil$.
8: Set $L_m = \ell$
9: A$(i, m-1, \ell+1)$ ▷ Left recursion
10: A$(m+1, j, \ell+1)$ ▷ Right recursion
11: **end if**
12: **end procedure**

the key(s) in $[i, j]$ at the minimum level in the optimal trees across all scenarios L^s (lines 5 and 6) and stores them in the ordered set S. If an odd number of keys share this minimum level, the algorithm selects the key to be stored at level ℓ as the median of the set S. Otherwise, it chooses the key closest to the median value (line 7).

We show that R-BST has competitive ratio logarithmic in k:

Theorem 2.4. *For every scenario s,* R-BST *constructs a BST of cost at most $\lceil \log_2(k+1) \rceil$ times the cost of the optimal BST L^s.*

We can implement R-BST so that it runs in time $O(kn^2)$, i.e., the run time is dominated by the time required to compute the optimal BSTs for each scenario.

Theorem 2.5. R-BST *can be implemented in time $O(kn^2)$.*

We conclude this section by showing that R-BST achieves the optimal competitive ratio.

Theorem 2.6. *There exists a collection of k scenarios, described by vectors $F^1, \ldots, F^k$, such that for every BST with level vector L, there exists a scenario s for which $\text{cost}(L, F^s) \geqslant \lceil \log_2(k+1) \rceil \text{cost}(L^s, F^s)$.*

3 Robust Huffman Trees

3.1 Background and Measures

In the standard version of the Huffman tree problem, we are given n keys (e.g., letters of an alphabet), along with a frequency vector (e.g., the frequency of each letter in said alphabet). The objective is to build a binary tree in which every key corresponds to a *leaf*, so as to minimize the inner product $\sum_i F_i L_i$, where L_i is the level of key i in the tree.

By labeling the edges leaving each inner node with $0, 1$, arbitrarily, we can associate a *codeword* to each key, namely the concatenation of the edge labels on the path from the root to the leaf corresponding to the key in question. In this setting, we assume that the levels of the tree start from 0. Hence, conveniently, the length of a codeword equals the level of the corresponding leaf. The resulting set of codewords is *prefix free*, in the sense that no codeword is a prefix of another codeword. The optimal tree can be computed in time $O(n \log n)$ (Huffman 1952).

We study Huffman coding in a robust setting in which we are given k n-dimensional frequency vectors $F^1, \ldots, F^k$, each describing a different scenario. We say that a level vector L is *valid* if there is a prefix-free code such that key i has a codeword of length L_i. In other words, for any valid level vector L, there is a Huffman tree with $|\{i : L_i = a\}|$ leaves on level a. The cost of L in a given scenario s is defined as the inner product $F^s \cdot L = \sum_i F^s_i L_i$.

We can analyze robust HTs using the same measures as for robust BSTs. For Huffman trees, in particular, we will rely on regret-based analysis, which establishes more refined performance guarantees than competitive ratio; c.f. Corollary 3.5 which shows that our results for regret essentially carry over to the competitive ratio.

3.2 Results

We begin by showing that finding an optimal robust HT is NP-hard, even in the case of only two scenarios. The proof differs substantially from the NP-hardness proof for robust BSTs (Theorem 2.3). This is due to the differences in the two settings: in a BST, keys are stored in each node, whereas in a HT, keys are stored only at leaf nodes. As a result, the reduction is technically more involved, and is from a problem that induces more structure, namely the EQUAL-CARDINALITY PARTITION problem (Garey and Johnson 1979).

Theorem 3.1. *The robust HT problem is NP-hard, even if $k = 2$. This holds for all three metrics, i.e., for minimizing the cost, the competitive ratio, or the regret.*

We propose and analyze an algorithm called R-HT for minimizing k-scenario regret. The idea is to aggregate the optimal trees for each scenario into a single HT. The algorithm initially computes optimal HTs for each scenario s, denoted as T^s (line 2). For each key $i = 1, \ldots, n$, it identifies the scenario s with the shortest code, denoted as $c^s_i \in \{0, 1\}^*$ (line 6). Next, it prepends exactly $\lceil \log_2 k \rceil$ bits to each code c^s_i, which represent the scenario s (lines 7 and 8). The algorithm generates the final HT by associating each key with a level equal to $\lceil \log_2 k \rceil + c^s_i$ (line 10). In line 11, "compactification" refers to a process which we describe informally, for simplicity. That is, while there is an inner node u of the HT with outdegree 1 (meaning it has a single descendant v) we contract the edge u, v.

Theorem 3.2. *Algorithm* R-HT *outputs a tree T with a valid level vector L of regret at most $\lceil \log_2 k \rceil$. That is, $F^s \cdot L \leqslant \min_{L^*} F^s \cdot L^* + \lceil \log_2 k \rceil$, for every scenario s.*

Proposition 3.3. *Algorithm* R-HT *can be implemented in time $O(kn \log n)$, i.e., its runtime is dominated by the time required to find optimal HTs for each scenario.*

Algorithm 3: R-HT

Input: k scenarios described by frequency vectors $F^1, \ldots, F^k$.
Output: A robust Huffman tree T.

1: **for all** scenarios s **do**
2: Let T^s be the Huffman tree with minimum cost for frequency vector F^s
3: **end for**
4: Let $\mathcal{C}$ be an empty set
5: **for all** keys i **do**
6: Let s be a scenario for which key i has the shortest code $c^s_i \in \{0, 1\}^*$
7: Let x be the encoding of the integer $s - 1$ with exactly $\lceil \log_2 k \rceil$ bits
8: Add xc^s_i to $\mathcal{C}$
9: **end for**
10: Build the Huffman tree T for the prefix free codewords $\mathcal{C}$
11: Compactify T
12: **return** T

The following result establishes a lower bound for our problem, and shows that R-HT is essentially best-possible.

Theorem 3.4. *There exists a set of k-scenarios for which no robust HT has regret smaller than $\lfloor \log_2 k \rfloor$.*

Corollary 3.5. *There exists an algorithm for k-scenario robust HTs of competitive ratio at most $\lceil \log k \rceil + 1$. Furthermore, no algorithm for this problem can have competitive ratio better than $\lfloor \log k \rfloor + 1$.*

4 Regret and Fairness in Binary Search Trees

In this section, we introduce the first study of fairness in BSTs, and demonstrate its connection to regret minimization in a *multiobjective* optimization setting. We begin with a motivating application. Consider a company that stores client information in a database structured as a BST. The clients are from either Spain or France, and the company would like to use a single database to store client data, instead of two, for simplicity of maintenance. Moreover, the company would like to treat customers of the two countries in a fair manner. Namely, the average cost for accessing clients from France should be comparable to that of accessing a database containing only these French clients, and similarly for Spanish clients.

We can formulate applications such as the above using a scenario-based regret-minimization framework over BSTs. Specifically, we can model the setting using two frequency vectors, one for each country. Each vector stores the probability of accessing a client, i.e., entry i is the probability of accessing client i. Since we treat all clients of a country equally, we are interested in tradeoffs between the average access costs of clients in the two countries, and can thus assume that the access probabilities are uniform. That is, if f denotes the number of French clients, then each such client has access probability $1/f$ in the frequency vector[1].

There are two important observations to make. First, unlike the approaches of Sections 2 and 3, the fairness setting

[1] In the application we discuss, a client can belong to one of two countries only. This is not a required assumption, but one we can make without loss of generality.

is inherently multi-objective. For example, in the above application, we are interested in the tradeoff between the total access costs of clients in the two countries. We will thus rely on the well-known concept of *Pareto-optimality* (Boyd and Vandenberghe 2004) that allows us to quantify such tradeoffs, as we will discuss shortly. Second, we will use the case of two scenarios for simplicity, however we emphasize that the setting and our results generalize to multiple scenarios, as we discuss at the end of the section.

We formalize our setting as follows. We are given two scenarios **0** and **1** over n keys $1, \ldots, n$. We denote by a and b the number of keys in scenario 0, and 1, respectively. We refer to keys of scenario 0 as the 0-*keys*, and similarly for 1-keys. We can describe compactly the two scenarios using a *binary string* $s \in \{0,1\}^n$, which specifies that key i belongs to scenario $s_i \in \{0,1\}$. Consider a BST T for this set of n keys, then the cost of key $i \in [n]$ is the level of the node in T that contains i. The 0-*cost* of T is defined as the total cost of all 0-keys in T, and the 1-cost is defined similarly.

To define the concept of regret, let $\mathrm{OPT}(m)$ denote the optimal cost of a binary tree over m keys, assuming a uniform key distribution. From (Knuth 1997, Sect 5.3.1, Eq. (3)),

$$\mathrm{OPT}(m) = (m+1)\lceil \log_2(m+1) \rceil - 2^{\lceil \log_2(m+1) \rceil} + 1. \quad (2)$$

Clearly, in every BST T, the 0-cost is at least $\mathrm{OPT}(a)$. We refer to the difference between the 0-cost of T and the quantity $\mathrm{OPT}(a)$ as the 0-*regret* of T. Thus, the 0-regret captures the additional cost incurred for searching 0-keys, due to the presence of 1-keys in T (1-regret is defined along the same lines). This notion allows us to establish formally the concept of fairness in a BST:

Definition 4.1. A BST for a string s is (α, β)-fair if it has 0-regret α and 1-regret β. We call (α, β) the *regret point* of the BST. We denote by $f(s, \alpha)$ the function that determines the smallest β such that there is a BST for s which is (α, β)-fair.

We say that a tree T dominates a tree T' if the regret point (α, β) of T dominates the regret point (α', β') of T' in the sense that $\alpha \leq \alpha'$ and $\beta \leq \beta'$ with one of the inequalities being strict. The Pareto-front is comprised by the regret points of all undominated trees.

4.1 Computing the Pareto Frontier

We now give an algorithm for computing the Pareto frontier. More precisely, we describe how to compute the function $f(s, \alpha)$. Note that f is non-increasing in α. The Pareto front is obtained by calling f for all $\alpha = 0, 1, \ldots$, until $f(s, \alpha) = 0$. We first need to bound the range of α.

Lemma 4.2. *Let α^* denote the smallest integer such that $f(s, \alpha^*) = 0$. It holds that $\alpha^* \leq a\lfloor \log_2(b+2) \rfloor$.*

Central to the computation of f is the notion of *loss*. For a given BST, we associate with each node, and for each scenario $c \in \{0,1\}$, a c-loss, such that the c-regret equals the total c-loss over all nodes. Informally, the c-loss at a node r is the increase of the c-cost due to the choice of r as the root of its subtree. Formally, we map $m_1, m_2 \in \mathbb{N}$ and $m_0 \in \{0,1\}$ to $\mathrm{loss}(m_1, m_0, m_2) = m_1 + m_0 + m_2 + \mathrm{OPT}(m_1) + \mathrm{OPT}(m_2) - \mathrm{OPT}(m_1 + m_0 + m_2)$, and recall that OPT is given by (2).

The interpretation of this definition is the following. Consider a BST for a string $s \in \{0,1\}^n$ with node r at its root. Let m_1 be the number of c in the left sub-string $s[1 : r-1]$, m_2 the number of c in the right sub-string $[r+1 : n]$, and m_0 the characteristic bit indicating whether s_r equals c. Then, if in both sub-trees scenario c has zero regret, then its overall cost is exactly $\mathrm{loss}(m_1, m_0, m_2)$.

We show how to compute the function f by dynamic programming. While one could use the approach of (Giegerich, Meyer, and Steffen 2004), which computes the Pareto-optimal regret points for all BSTs for all sub-strings of s, we propose a somewhat different approach that has the same time complexity and is easier to implement.

The empty string $s = \varepsilon$ constitutes the base case for which we have $f(\varepsilon, \alpha) = 0$. For $s \neq \varepsilon$ of length $n \geq 1$ we have

$$f(s, \alpha) = \min_r \min_{\alpha_1, \alpha_2} (f(s[1, r-1], \alpha_1) + \mathrm{loss}(b_1, b_0, b_2) + f(s[r+1, n], \alpha_2)),$$

where the root r in the outer minimization ranges in $[1, n]$ and separates the string to a left sub-string $s[1 : r-1]$, a root s_r, and a right sub-string $s[r+1, n]$. For a fixed r, the value a_1 is the number of 0s in the left sub-string, whereas a_2 is the number of 0s in the right sub-string, a_0 is the indicator bit for $s_r = 0$, and b_1, b_0, b_2 are similarly defined for scenario **1**. The inner minimization optimizes over all partitions α_1, α_2 of the allowed bound on the 0-regret for the left and right sub-trees, such that $\alpha = \alpha_1 + \mathrm{loss}(a_1, a_0, a_2) + \alpha_2$.

The correctness of the algorithm follows from the fact that any BST for s with 0-regret at most α is defined by a root and a partition of the remaining 0-regret $\alpha - \mathrm{loss}(a_1, a_0, a_2)$, and is composed recursively by a left and right sub-tree. The algorithm's running time is $O(n^3(\alpha^*)^2)$, which simplifies to $O(n^5 \log^2 n)$ by Lemma 4.2.

We emphasize that the dynamic programming approach can be generalized to $k \geq 2$ scenarios. In this general setting, a regret vector has dimension k, and in the function f we fix $k-1$ dimensions and optimize the last one. Hence, we have $O(n^2(n \log n)^{k-1}) = O(n^{1+k} \log^{k-1} n)$ variables, each being a minimization over $O(n(n \log n)^{k-1})$ choices, which yields a time complexity of $O(n^{1+2k} \log^{2k-2} n)$.

5 Computational Experiments

5.1 Robust BSTs and HTs

We report computational experiments on the robust versions of BSTs and HTs from Sections 2 and 3. We used open data from (Wikipedia 2024). Specifically, we chose ten European languages[2] as corresponding to ten different scenarios, based on the frequency of each letter in the corresponding language. We restrict to the English alphabet of 26 letters, ignoring other letters or accents for simplification, but normalizing the frequencies to 1. For example, the most frequent letter in English is `e` with 12.7%, whereas in Portuguese the letter `a` is used more often with frequency 14.6%. To evaluate our algorithms, we provide a mixed integer linear

[2]Danish, Dutch, English, Finnish, French, German, Italian, Portuguese, Spanish and Swedish.

	Binary Search Tree		Huffman Tree	
	Optimal	R-BST	Optimal	R-HT
cost	3.389	3.940	4.271	4.425
comp. ratio	1.047	1.215	1.038	1.091
regret	0.151	0.680	0.155	0.364

Table 1: Performance comparison of the various algorithms.

program (MILP) formulation for the problems. This allows us to compute optimal trees with the help of commercial MILP solvers, such as GUROBI. We give the MILP for cost-minimization in robust BSTs, but we note that minimizing the competitive ratio and the regret follow along the same lines, by only changing the objective function accordingly. The range of indices is $i, j, \ell \in [1, \ldots, n]$.

$$\min \quad C \tag{3}$$

$$\text{subj. to} \quad \forall i : \sum_{\ell} x_{\ell,i} = 1 \tag{4}$$

$$\forall i, j, \ell : \sum_{r=i+1}^{j-1} \sum_{u=1}^{l-1} x_{u,r} \geqslant x_{\ell,i} + x_{\ell,j} - 1 \tag{5}$$

$$\forall s : \sum_{\ell} \sum_{i} F_i^s \cdot \ell \cdot x_{\ell,i} \leqslant C \tag{6}$$

$$\forall i, \ell : \; x_{\ell,i} \in \{0, 1\} \tag{7}$$

Here, $x_{\ell,i}$ is that $x_{\ell,i} = 1$ if and only if key i is assigned to level ℓ. Constraint (4) ensures that every key is assigned to exactly one level. Constraint (5) ensures that the resulting level vector satisfies Definition 2.1. Last, constraint (6) together with the objective (3) guarantee that C is the maximum tree cost over all scenarios. The MILP for robust HTs can be obtained similarly.

GUROBI solved the MILPs for our experimental setting in times ranging from 7 to 15 seconds on a standard laptop. In contrast, R-BST and R-HT run in less than 0.1 seconds, using a Python implementation. Table 1 summarizes the results of the experiments. As expected, the empirical performance is better than the worst-case guaranties of Theorems 2.4 and 3.2. This is due to the fact that real data do not typically reflect adversarial scenarios (compare, e.g., to the adversarial constructions of Theorem 2.6 and Theorem 3.4).

5.2 Pareto-Optimality and Fairness

We report experiments on our Pareto-optimal algorithm of Section 4, denoted by PO. The universe of all possible keys is represented by the names of cities from 10 different countries (the same counties as in Section 5.1). From this data, a string s of length $2n$, for some chosen n, is generated by selecting two of the above countries (which we call country 0 and country 1) and the n largest in population cities from each country. Then, s is obtained by sorting lexicographically (i.e., alphabetically) the $2n$ cities and by setting $s_i \in \{0, 1\}$, depending on whether the i-th city in s belongs to country 0 or 1.

We run algorithm PO and found the regret-based Pareto front for each string s generated as above, choosing $n = 30$.

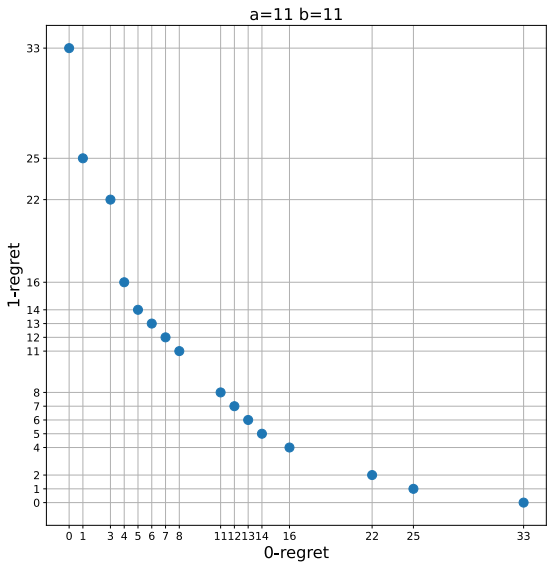

Figure 2: The Pareto front for strings with $a = 11, b = 11$.

From the results, we observed that for every s, there exists a BST whose 0-regret and 1-regret are *both* bounded by n. Specifically, for every s, we were able to find a tree with a 0-regret of 27 and 1-regret of 28, as well as a tree with a 0-regret of 28 and a 1-regret of 27. These regret numbers were obtained for all $\binom{10}{2}$ country pairs.

Furthermore, for all $a, b \in \{0, \ldots, 11\}$, we generated all strings s of size $a + b$, i.e., a keys from scenario **0** (country 0) and b keys from scenario **1** (country 1). Figure 2 illustrates our findings for the case $a = b = 11$. Here, a point (α, β) signifies that we can find a BST of 0-regret α and 1-regret β for *any* string with 11 0s and 11 1s. This can be accomplished by running algorithm PO on all possible such strings. We obtained the same conclusion for all strings in which $a \leq 11$ and $b \leq 11$.

Remark 5.1. The experimental results suggest that for a string s consisting of a 0s and b 1s, there exists a BST of 0-regret at most a and 1-regret at most b.

6 Conclusion

We introduced the study of scenario-based robust optimization in data structures such as BSTs, and in data coding via Huffman trees. We gave hardness results, and theoretically optimal algorithms for a variety of measures such as competitive ratio, regret and Pareto-optimality. Our work also established connections between fairness and multi-objective regret minimization. Future work will address other important data structures, such as B-trees and quad-trees, as well as approximation algorithms to the NP-hard problem of minimizing the worst-case tree cost. Our approaches address a fundamental issue: the tradeoff between cost and frequency of operations, which can be of use in many other practical domains, such as inventory management in a warehouse.

Acknowledgements

This work was supported by the grant ANR-23-CE48-0010 PREDICTIONS from the French National Research Agency (ANR).

References

Angelopoulos, S.; Dürr, C.; Elenter, A.; and Melidi, G. 2024. Scenario-Based Robust Optimization of Tree Structures. *arXiv preprint arXiv:2408.11422*.

Arsenis, M.; and Kleinberg, R. 2022. Individual Fairness in Prophet Inequalities. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, 245–245.

Ben-Tal, A.; El Ghaoui, L.; and Nemirovski, A. 2009. *Robust optimization*, volume 28. Princeton university press.

Blum, A.; and Mansour, Y. 2007. Learning, Regret Minimization, and Equilibria. In N, N.; T, R.; E, T.; and VV, V., eds., *Algorithmic Game Theory*, 79–102. Cambridge University Press.

Boffa, A.; Ferragina, P.; and Vinciguerra, G. 2022. A learned approach to design compressed rank/select data structures. *ACM Transactions on Algorithms (TALG)*, 18(3): 1–28.

Booth, A. D.; and Colin, A. J. 1960. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4): 327–334.

Borodin, A.; and El-Yaniv, R. 2005. *Online computation and competitive analysis*. cambridge university press.

Boyd, S.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Buchbinder, N.; Jain, K.; and Singh, M. 2009. Secretary problems and incentives via linear programming. *ACM SIGecom Exchanges*, 8(2): 1–5.

Cao, X.; Chen, J.; Chen, L.; Lambert, C.; Peng, R.; and Sleator, D. 2022. Learning-augmented b-trees. *arXiv preprint arXiv:2211.09251*.

Ferragina, P.; and Vinciguerra, G. 2020. *Learned Data Structures*, 5–41. Cham: Springer International Publishing. ISBN 978-3-030-43883-8.

Fu, C.; Seo, J. H.; and Zhou, S. 2024. Learning-Augmented Skip Lists. *arXiv preprint arXiv:2402.10457*.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and intractability*. W.H.Freeman & Co Ltd. ISBN 978-0716710455.

Giegerich, R.; Meyer, C.; and Steffen, P. 2004. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3): 215–263.

Gilenson, M.; and Shabtay, D. 2021. Multi-scenario scheduling to maximise the weighted number of just-in-time jobs. *Journal of the Operational Research Society*, 72(8): 1762–1779.

Hibbard, T. N. 1962. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM (JACM)*, 9(1): 13–28.

Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9): 1098–1101.

Kasperski, A.; Kurpisz, A.; and Zielinski, P. 2012. Approximating a two-machine flow shop scheduling under discrete scenario uncertainty. *Eur. J. Oper. Res.*, 217(1): 36–43.

Kasperski, A.; Kurpisz, A.; and Zieliński, P. 2015. Approximability of the robust representatives selection problem. *Operations Research Letters*, 43(1): 16–19.

Kasperski, A.; and Zieliński, P. 2015. Complexity of the robust weighted independent set problems on interval graphs. *Optimization Letters*, 9: 427–436.

Knuth, D. E. 1971. Optimum binary search trees. *Acta informatica*, 1: 14–25.

Knuth, D. E. 1997. *The art of computer programming*, volume 3. Pearson Education.

Kraska, T.; Beutel, A.; Chi, E. H.; Dean, J.; and Polyzotis, N. 2018. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*, 489–504.

Lechowicz, A.; Sengupta, R.; Sun, B.; Kamali, S.; and Hajiesmaili, M. 2024. Time Fairness in Online Knapsack Problems. In *The Twelfth International Conference on Learning Representations*.

Lin, H.; Luo, T.; and Woodruff, D. 2022. Learning augmented binary search trees. In *International Conference on Machine Learning*, 13431–13440. Proceedings of Machine Learning Research.

Mankowski, M.; and Moshkov, M. 2020. Dynamic programming bi-criteria combinatorial optimization. *Discrete Applied Mathematics*, 284: 513–533.

Manthey, B.; and Reischuk, R. 2007. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3): 292–315.

Martinez, N.; Bertran, M.; and Sapiro, G. 2020. Minimax pareto fairness: A multi objective perspective. In *International conference on machine learning*, 6755–6764. PMLR.

Martinez, N. L.; Bertran, M. A.; Papadaki, A.; Rodrigues, M.; and Sapiro, G. 2021. Blind pareto fairness and subgroup robustness. In *International Conference on Machine Learning*, 7492–7501. PMLR.

Mastrolilli, M.; Mutsanas, N.; and Svensson, O. 2013. Single machine scheduling with scenarios. *Theoretical Computer Science*, 477: 57–66.

McCarthy, S. M.; Vayanos, P.; and Tambe, M. 2017. Staying Ahead of the Game: Adaptive Robust Optimization for Dynamic Allocation of Threat Screening Resources. In *IJCAI*, 3770–3776.

Moffat, A. 2019. Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4): 1–35.

Nagaraj, S. 1997. Optimal binary search trees. *Theoretical Computer Science*, 188(1): 1–44.

Nanongkai, D.; Sarma, A. D.; Lall, A.; Lipton, R. J.; and Xu, J. 2010. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2): 1114–1124.

Patel, D.; Khan, A.; and Louis, A. 2021. Group fairness for knapsack problems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 2, 989–997.

Shabtay, D.; and Gilenson, M. 2023. A state-of-the-art survey on multi-scenario scheduling. *European Journal of Operational Research*, 310(1): 3–23.

Sleator, D. D.; and Tarjan, R. E. 1985. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3): 652–686.

Wikipedia. 2024. Letter frequency. https://en.wikipedia.org/wiki/Letter_frequency. [Online; accessed 09-February-2024].

Windley, P. F. 1960. Trees, forests and rearranging. *The Computer Journal*, 3(2): 84–88.

Xie, M.; Wong, R. C.-W.; and Lall, A. 2020. An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. *The VLDB Journal*, 29(1): 147–175.

Zeynali, A.; Kamali, S.; and Hajiesmaili, M. 2024. Robust Learning-Augmented Dictionaries. *arXiv preprint arXiv:2402.09687*.