



Algorithms Under Uncertainty : Robust Methods and Decision-Theoretic Evaluation

Georgii Melidi

► To cite this version:

Georgii Melidi. Algorithms Under Uncertainty : Robust Methods and Decision-Theoretic Evaluation. Data Structures and Algorithms [cs.DS]. Sorbonne Université, 2025. English. ⟨NNT : 2025SORUS226⟩. ⟨tel-05351852⟩

HAL Id: tel-05351852

<https://theses.hal.science/tel-05351852v1>

Submitted on 6 Nov 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization



Thèse de doctorat en informatique
de Sorbonne Université

École Doctorale Informatique, Télécommunications et
Électronique

Algorithms Under Uncertainty: Robust Methods and Decision-Theoretic Evaluation

Georgii Melidi

présentée et soutenue publiquement
pour obtenir le grade de
Docteur en Sciences Informatique

Date provisoire de soutenance: 25 septembre 2025

Président du jury: Adi Rosén

Directeurs de thèse

Spyros Angelopoulos

CNRS, International Laboratory on Learning Systems, Montreal, Canada

Christoph Dürr

CNRS, Sorbonne Université, LIP6, France

Rapporteurs

Antonios Antoniadis

Discrete Mathematics and Mathematical Programming group, University of Twente, Enschede, The Netherlands

Johanne Cohen

CNRS, Laboratoire LISN, Université Paris-Saclay, France

Examineurs

Cristina Bazgan

Université Paris Dauphine, LAMSADE, France

Adi Rosén

CNRS, IRIF, Université Paris Cité, France

Patrice Perny

Sorbonne Université, LIP6, France

Michèle Sebag

CNRS, Laboratoire LISN, Université Paris-Saclay, France

Abstract

In many algorithmic problems involving sequential decision-making, uncertainty is a central challenge. Input items may arrive over time, the input distribution may be unknown or partially known, and predictions may be available but not always accurate. Addressing these challenges requires new algorithmic models and techniques which are suited for partial-information settings, can capture uncertain or scenario-based inputs, and can support analysis across varying levels of prediction accuracy.

This thesis studies two main algorithmic settings that involve uncertainty. The first setting is *scenario-based* optimization, in which the algorithm is faced with a set of input scenarios, each representing a distinct possibility that a certain input materializes. We present the first study of fundamental data structures such as *Binary Search Trees* and *Huffman Trees* under scenario-based optimization, where the goal is to obtain a single tree that performs well across all scenarios. Using theoretical performance metrics such as the competitive ratio and the regret, we give NP-hardness proofs and near-tight approximations, as well as a validating experimental analysis. We also show that this framework can help us capture the notion of *fairness*, in the context of data structures.

Furthermore, we show how to apply the scenario-based approach to the classic *prophet inequality problem*, where the input distribution is known to belong to a given set of distributions. We design stopping rules that adapt to observed values and achieve optimal worst-case guarantees, while simultaneously demonstrating efficient empirical performance.

The second part of this thesis focuses on *learning-augmented* algorithms, in which the online algorithm is enhanced with a potentially erroneous *prediction*. Here, we study approaches based on *decision theory* that allow us to rigorously quantify and compare the performance of algorithms as the prediction quality varies. Specifically, we introduce distance-based and risk-aware metrics that allow for a refined performance analysis and help select the most suitable algorithm from a broad class of candidates.

Résumé

Dans de nombreux problèmes algorithmiques liés à la prise de décision séquentielle, l'incertitude constitue un défi central. Les éléments d'entrée peuvent arriver de manière progressive, la distribution sous-jacente peut être inconnue ou partiellement connue, et des prédictions peuvent être disponibles sans pour autant être toujours fiables. Relever ces défis nécessite de nouveaux modèles algorithmiques et des techniques adaptées aux environnements à information partielle, capables de représenter des données incertaines ou dépendant de scénarios, et permettant une analyse selon différents niveaux de précision des prédictions.

Cette thèse étudie deux cadres algorithmiques principaux intégrant l'incertitude. Le premier est celui de l'*optimisation basée sur des scénarios*, où l'algorithme fait face à un ensemble de scénarios représentant différentes réalisations possibles des données d'entrée. Nous présentons la première étude de structures de données fondamentales telles que les *arbres de recherche binaires* et les *arbres de Huffman* dans ce cadre, avec pour objectif de construire un arbre unique offrant de bonnes performances sur l'ensemble des scénarios. En utilisant des métriques théoriques telles que le *rapport de compétitivité* et le *regret*, nous démontrons des résultats de complexité (preuves de NP-difficulté) et proposons des algorithmes d'approximation quasi optimaux, accompagnés d'une validation expérimentale. Nous montrons également que ce cadre permet de formaliser la notion d'*équité* dans l'usage des structures de données.

Nous illustrons en outre l'applicabilité de cette approche au problème classique de l'*inégalité du prophète*, dans lequel la distribution d'entrée est supposée appartenir à un ensemble donné. Nous concevons des règles d'arrêt qui s'adaptent aux valeurs observées et garantissent des performances optimales dans le pire des cas, tout en présentant une efficacité empirique satisfaisante.

La seconde partie de la thèse est consacrée aux *algorithmes en ligne augmentés par apprentissage* (*learning-augmented algorithms*), dans lesquels l'algorithme dispose d'une *prédiction* pouvant être erronée. Nous étudions des approches fondées sur la *théorie de la décision*, permettant d'évaluer rigoureusement les performances des algorithmes en fonction de la qualité de la prédiction. En particulier, nous introduisons des métriques basées sur la distance et la gestion du risque, qui permettent une analyse plus fine et facilitent le choix de l'algorithme le plus adapté parmi une large classe de candidats.

Acknowledgments

These past three years at LIP6 have been a formative and rewarding period of my life, both scientifically and personally. I am very grateful for the opportunity to do my PhD under the supervision of Spyros Angelopoulos and Christoph Dürr. Spyros, thank you for your patience and support. The time we spent discussing research was always productive and helped me understand what careful and high-quality scientific work looks like. You taught me how to organize ideas clearly, how to write papers with precision, and how to keep focus on what truly matters. Even during the past year, when we were in different places, we stayed in close and regular contact, and I deeply appreciate that. Christoph, it has been a real pleasure to work with you during these years. I was always impressed by your creativity and your ability to think in an out-of-the-box way. Your drawings helped to make complex ideas easy to explain and understand. I learned a lot from the way you think and from your original and visual approach to research. I also really appreciated the times we went to work outside the university. Changing the setting and working in a café together always brought a fresh perspective and made research feel even more enjoyable.

Before starting my PhD, I completed both my Bachelor's and Master's studies at Novosibirsk State University under the supervision of Adil Ilyasovich Erzin. Adil Ilyasovich, thank you for your guidance during those years. You helped shape my academic interests and gave me a strong foundation in research. I would also like to thank Alexander Veniaminovich Kononov. You were the one who encouraged me to consider doing a PhD abroad and specifically suggested applying to LIP6. Your advice played an important role in the path I eventually chose. I am truly grateful to both of you for your support and belief in me at the early stages of my academic journey.

At LIP6, I had the chance to be surrounded by a welcoming and supportive community. I want to thank everyone I shared this time with: Denis Antipov, André Conde Vázquez, François Clément, Gianluca Covini, Catalin-Viorel Dinu, Duri Andrea Janett, Martin Durand, Océane Fourquet, Alexis Guigal, Miguel González-Duque, Yihang Lu, Maher Mallem, Sasa Mladenovic, Daniel Nowak-Assis, Emile Royer, Dimitri Rusin, Mara Santarelli, Maria Laura Santoni, Diederick Vermetten, and Michalis Xeferis. Thank you for all the discussions, help, jokes, board games, and activities outside the lab that made this time both productive and fun. I am also

grateful to the permanent members of the team, especially Carola Doerr, Bruno Escoffier, and Evripidis Bampis, for their kindness and their willingness to help whenever it was needed. Your presence made the lab a truly enjoyable place to work.

Thank you to all members of my jury, both reviewers and examiners, for taking the time to read and evaluate my thesis. I sincerely appreciate your attention, your engagement, and your role in bringing this work to its conclusion.

Beyond research, I want to thank my beloved Marina for being by my side throughout these years. Your support means the world to me. I am so grateful for your care, for believing in me when I doubted myself, and for always finding the right words to keep me going. Thank you for gently pushing me forward, for your patience, and for all the quiet strength you shared with me. I am deeply grateful for everything, and I hope you know how much you mean to me.

Finally, I want to thank my family. To my mother, Ekaterina, thank you for your endless belief in me, your patience, and your support. I always feel how deeply you care, sometimes even more than I do myself. To my father, Evstafiy, thank you for your thoughtful advice, steady support, and quiet confidence that I would succeed. I am also grateful to my brother Miron and my sister Kseniia. You are the best, and without you, this journey would have been much harder. Thank you for always being there for me and for your constant encouragement. I also want to thank my grandparents for their warmth, care, and love, which I have always felt no matter the distance. And of course, to all my friends — thank you for being close, even when far away, and for always standing by my side.

Contents

Abstract	v
Résumé	vii
Acknowledgments	ix
Contents	xi
1 Introduction	1
1.1 Objectives of the Thesis	4
1.2 Contributions of the Thesis	5
1.2.1 Scenario-Based Robust Optimization of Tree Structures	6
1.2.2 Prophet Inequalities with Scenarios	7
1.2.3 Decision-Theoretic Approaches in Learning-Augmented Algorithms	7
1.3 Outline of the Thesis	8
I Background	11
2 Models and Preliminaries	13
2.1 Online Algorithms	13
2.1.1 Competitive Analysis	13
2.1.2 Learning-Augmented Algorithms	16
2.1.3 Further problems and applications	20
2.2 Robust Optimization	26
2.2.1 Scenario-Based Robust Optimization	27
2.3 Tree Data Structures	28
2.3.1 Binary Search Trees	29
2.3.2 Huffman Trees	31

II	Contributions	35
3	Scenario-Based Robust Optimization of Tree Structures	37
3.1	Introduction	37
3.1.1	Contribution	38
3.1.2	Further Related Work	39
3.2	Robust Binary Search Trees	41
3.2.1	Background and Measures	41
3.2.2	Results	43
3.3	Robust Huffman trees	47
3.3.1	Background and Measures	47
3.3.2	Results	48
3.4	Regret and Fairness in Binary Search Trees	53
3.4.1	Computing the Pareto Frontier	56
3.5	Computational Experiments	60
3.5.1	Robust BSTs and HTs	60
3.5.2	Pareto-Optimality and Fairness	62
3.6	Relation to the Entropy	65
3.7	Conclusion	67
4	Prophet Inequalities with Scenarios	69
4.1	Introduction	69
4.1.1	Further Related Work	69
4.2	Model	70
4.3	A Tight Upper Bound on the Competitive Ratio	71
4.4	A Baseline Strategy	73
4.5	Observation-Based Strategies	74
4.5.1	Commitment	75
4.5.2	Belief Update	76
4.6	Belief-Driven Decision Rules	77
4.7	Experimental Evaluation	79
4.7.1	Datasets	79
4.7.2	Results	80
4.8	Conclusion	83
5	Decision-Theoretic Approaches in Learning-Augmented Algorithms	85
5.1	Introduction	85
5.1.1	Contributions	86

5.1.2	Other related work	87
5.2	Decision-theoretic models	89
5.2.1	Distance-based analysis	89
5.2.2	Risk-based analysis	90
5.3	Application in ski rental	91
5.3.1	Distance Measures	92
5.3.2	Risk-Based Analysis	96
5.4	Application in 1-max search	99
5.4.1	Distance measures	100
5.4.2	Risk-Based Analysis	106
5.5	Application in contract scheduling	109
5.5.1	Distance measures	109
5.5.2	Risk-based analysis	114
5.6	Experimental Evaluation	117
5.6.1	Evaluation of Ski Rental Algorithms	117
5.6.2	Evaluation of 1-Max Search Algorithms	119
5.6.3	Evaluation of Contract Schedules	125
5.7	Conclusion	127
6	Conclusions & Outlook	129
	Bibliography	133

Computer Science has been guided by the goal of finding efficient solutions to computationally difficult problems. In the early days, much of the focus was on problems in which the entire input was well-defined and known in advance, and the goal was to compute an efficient solution, in terms of the runtime or the quality of the output. Knowing the input helped not only with the design of the algorithms, but more importantly with the theoretical analysis.

To illustrate with an example, consider one of the earliest and most studied problems, namely *sorting*. Sorting is a fundamental computational problem, with many real-world applications, and has long served as a benchmark for algorithm design. Sorting algorithms are evaluated on metrics such as correctness, running time, and memory usage. Concretely, algorithms such as Merge Sort and Quick Sort are well understood in both theory and practice [Cor+09]. Their success comes from the fact that we can analyze them precisely and compare them using well-defined measures.

However, many situations in the real world do not fit the ideal model in which the input is completely known in advance. In practice, the setting often involves some degree of *uncertainty*. Concretely, the algorithm may have to make decisions step by step, without knowing what will happen next; in other words, the input is not known ahead of time. Such settings necessitate a new model that accounts for inputs that appear *online*, i.e., piece-by-piece, and the algorithm must act immediately, based only on the information seen so far.

A good way to understand this is by thinking of the game Tetris. In Tetris, pieces appear on the screen one at a time, and the player must decide where to place each piece without knowing which shape will come next. The goal is to clear as many rows as possible by placing pieces efficiently. Once placed, the decision cannot be changed. This is conceptually similar to how online algorithms work: they receive one piece of information at

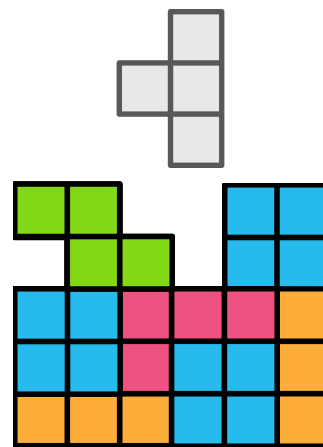


Figure 1.1: Online decision-making as in Tetris

a time, on which they must act without the option to look ahead or undo past decisions.

In these online settings, measures such as running time do not capture the full complexity of the problem. However, the concept of worst-case evaluation is still applicable. Specifically, in their seminal work [ST85a], Sleator and Tarjan introduced *competitive* analysis, which compares the performance of an online algorithm to that of an optimal offline algorithm that knows the full input in advance. Informally, the measure captures the multiplicative “loss” incurred by the algorithm due to the online nature of the setting. This is formalized in the concept of the *competitive ratio*.

► **Definition 1.1.** Given a cost-minimization problem and a deterministic online algorithm ALG, the competitive ratio of ALG is defined as:

$$CR(\text{ALG}) = \sup_{\sigma} \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)}$$

where $\text{ALG}(\sigma)$ denotes the cost of ALG on request sequence σ , and $\text{OPT}(\sigma)$ is the cost of an optimal offline algorithm. ◀

Competitive analysis, and the study of online algorithms, more generally, has evolved into a prolific field of Theoretical Computer Science, with several applications in many other fields. For a comprehensive treatment, we refer to the books [BE98; FW98]. We will discuss several examples of well-studied problems in Section 2.1.3.

Although the competitive ratio gives strong worst-case guarantees, it does not take into account any prior knowledge or patterns that might be present in the input. In real applications, we often have access to historical data or forecasts. This has motivated recent efforts to go *beyond the worst-case analysis* by incorporating predictions into the online algorithms. We discuss such models in the next section.

Learning-Augmented Online Computation

While many real-world problems involve uncertainty, in many cases we also have access to useful patterns or data that reflect how the input behaves. In recent years, there has been growing interest in using such information to improve the performance of online algorithms. This has led to the *learning-augmented* framework, where algorithms receive *predictions* about the future input, for instance produced by a machine learning model trained on past data.

The predictions may be helpful, but they are not guaranteed to be correct. The main challenge is to design algorithms that benefit from accurate predictions without failing when the predictions are wrong. This idea brings together the strengths of classic theoretical, and worst-case analysis with the insights obtained from the predictions. The goal is not to replace worst-case guarantees altogether, but rather to extend it to settings where some prior information is available.

A central part of this approach is understanding how the performance of an algorithm changes as the predictions become better or worse. A good algorithm should be able to use accurate predictions to make better decisions, but also continue to perform well when the predictions are not reliable. Finding the right balance between trusting the predictions and managing the prediction errors is one of the main challenges in this area.

The learning-augmented framework was first studied explicitly in the work of Lykouris and Vassilvitskii [LV21], and has since been applied to a wide range of problems. It should be noted that online models that incorporate external information have been studied, notably within the file of *advice complexity* [Böc+09; Boy+17; DKP09; Eme+11]. Here, the algorithm receives perfect hints from an ideal, error-free oracle, and the objective is to quantify the tradeoff between the competitive ratio and the amount of information (in terms of bits) provided by the oracle. In contrast, in the learning-augmented setting, the predictions may contain errors, and the algorithm must be designed with a certain degree of robustness against them.

Incorporating errors makes the model more relevant in practice, but introduces new questions and challenges about how to evaluate algorithms, and how to compare their performance under different prediction scenarios. Unlike in the standard competitive setting, the performance cannot be captured by a single metric or assessed with respect to a single goal. Instead, we often have to consider several aspects at once, such as how the algorithm behaves when predictions are perfect, how much it degrades when predictions are poor, and how smoothly it transitions between these two cases. This has led to a variety of evaluation methods and comparison techniques. As a result, the literature has grown rapidly, with hundreds of papers studying different problems, models, and objectives. An organized list of recent contributions in this area can be found in the online repository [LM25].

Robust Optimization

The second main direction we consider in this thesis is *robust optimization*. In this field the focus is on finding solutions to optimization problems which can provably

tolerate a certain degree of uncertainty. While machine learning tries to leverage information about the future, robust optimization takes a more cautious approach: it assumes that uncertainty is unavoidable and aims to find solutions that work well in the worst case.

The core idea in robust optimization is to model uncertainty explicitly by means of a set of possible *scenarios*. Namely, each scenario represents a different realization of unknown input data. The goal is to find a solution that performs well across all scenarios and, in particular, minimizes the cost under the worst-case scenario. This is especially useful when predictions are unreliable, or when the consequences of a bad decision are too costly.

Robust optimization has been used in many areas where uncertainty plays a major role, such as finance, supply chains, and energy systems. In supply chain management, for example, it helps decide how much inventory to keep when future demand is not known [BGN09]. In portfolio selection, it helps build investment plans that remain stable even when the market behaves unpredictably [GI03]. For an in-depth discussion, we refer to the book [BGN09].

From a theoretical point of view, robust optimization provides a way to extend the standard models to account for uncertainty in a clear and structured manner. A common method is to describe several possible inputs as scenarios, and then search for solutions that perform well across any one among them. In many cases, the goal is to minimize the worst-case cost among the scenarios. This approach is known as *scenario-based* robust optimization and has been studied in various discrete optimization problems, including network design, location planning, and scheduling [BS07; GMT14]. Although rooted in Operations Research, which very often focuses on probabilistic analyses, this approach incorporates worst-case analysis, which is an essential aspect in Theoretical Computer Science. That is, the algorithms and decision rules are evaluated not on their average performance, but on how well they perform under the most difficult or uncertain conditions.

1.1 Objectives of the Thesis

This thesis aims to contribute to the study of online algorithms and related areas, by focusing on situations where there is uncertainty in the input. The goal is to better understand how to design and evaluate algorithms when the future is not fully known, using ideas from both prediction models and robust optimization. The main objectives include:

- **Scenario-Based Analysis:** Here, we focus on two well-studied problems, namely the design and analysis of data structures such as binary search

trees, and optimal stopping problems such as prophet inequalities. While both problems are natural candidates for a study under (scenario-based) robust optimization, nothing has been known prior to this work. The main consideration is how to design data structures and stopping rules that are robust to uncertainty, and evaluate the resulting solutions both theoretically and experimentally.

- **New Evaluation Metrics for Learning-Augmented Algorithms:** In many well-studied problems, there exists a class of algorithms with similar, or even identical performance. This raises the need for refined design and analysis criteria to meaningfully distinguish among candidate algorithms and select the “best-possible”. This is a situation that is conceptually similar to the study of alternative measures in online computation, when competitive analysis does not yield satisfactory results [ST85a], and new approaches are required [DL05; HV12].

In a nutshell, our aim is to show that going beyond the standard models leads to a better understanding of how to make reliable decisions in uncertain environments.

1.2 Contributions of the Thesis

The results presented in this thesis are based on the following publications and preprints (the authors are listed in alphabetical order):

- **Scenario-Based Robust Optimization of Tree Structures**
Spyros Angelopoulos, Christoph Dürr, Alex Elenter, and Georgii Melidi.
Presented at the 39th AAAI Conference on Artificial Intelligence (AAAI), 2025 [Ang+25].
This paper was selected for oral presentation.
- **Decision-Theoretic Approaches in Learning-Augmented Algorithms**
Spyros Angelopoulos, Christoph Dürr, and Georgii Melidi.
Preprint currently under review [ADM25].
- **Prophet Inequalities with Scenarios**
Spyros Angelopoulos, Christoph Dürr, and Georgii Melidi.
Work in progress.

In the following sections, we give an overview of the contributions, their context, and their significance.

1.2.1 Scenario-Based Robust Optimization of Tree Structures

This part of the thesis initiates a novel study of tree structures under the framework of scenario-based robust optimization, focusing on Binary Search Trees (BSTs) and Huffman Trees. Both structures are fundamental in the fields of data management and coding, and have been traditionally optimized based on known frequency distributions of keys. We study a more general setting, in which there are k different possibilities about the keys that are to be stored, each defined as a distinct frequency distribution. Our goal is to compute a *single* tree that performs optimally across all scenarios.

To measure the performance of tree structures under this general setting, we apply two metrics: the competitive ratio and the *regret*. The competitive ratio provides a multiplicative comparison of the cost of a constructed tree to that of the least-costly tree under any scenario, while the regret offers an additive comparison, presenting an alternative angle for evaluation.

For Binary Search Trees, we show that the problem of finding the optimal tree is NP-hard under either of the performance metrics. We complement this negative result with an algorithm that finds a BST of competitive ratio $\lceil \log_2(k + 1) \rceil$, where k denotes the number of scenarios. We also establish the optimality of this ratio, by showing that it is tight on a worst-case collection of scenarios. This study significantly advances the knowledge of Binary Search Trees (BSTs), since extensive previous work focused on single inputs (hence a single scenario). We obtain similar results, both in terms of hardness and approximations, for Huffman Trees. Here, we show that the problem is once again NP-hard, and we introduce an algorithm that attains a regret of $\lceil \log_2 k \rceil$, supported by a proven near-optimal lower bound of $\lfloor \log_2 k \rfloor$.

Additionally, we establish the first connection between regret-based optimization and *fairness* in data structures. Specifically, we consider a setting in which several users have access to a data structure, such as a BST. Each user has its own pattern of accessing data, described as a distinct, uniform distribution over a subset of the keys. The objective is to find a single tree that optimizes the tradeoff on the regret of the different users.

In addition to the theoretical results, we present an experimental evaluation of our algorithms, which supports the theoretical analysis. To this end, we propose mixed integer linear program formulations so as to compute the optimal trees.

1.2.2 Prophet Inequalities with Scenarios

This part of the thesis extends the prophet inequality problem to a scenario-based setting, where the goal is to make decisions under distributional uncertainty. In the standard model, a decision-maker observes a sequence of n *values*, which are the outcomes of n non-negative, independent random variables, and must select one value irrevocably. The objective is to choose a value that approximates the ideal reward of a *prophet*, who knows the entire sequence in advance, while the algorithm has access only to the *distributions* of the random variables. It is known that a reward of at least half the expected maximum (prophet) can be achieved, and this bound is tight.

We introduce a model in which the distribution is not fully known, but instead belongs to one of k possible scenarios. Each scenario defines a distinct, n -dimensional vector of distributions, but the decision-maker does not know which specific vector (i.e., which specific scenario) applies. Our task is to select a single stopping rule that performs well across all possible scenarios.

We first provide a simple stopping rule that achieves an approximation ratio of $1/(2k)$, and prove that this is optimal in the worst case. While this stopping rule is theoretically optimal, it is tailored to worst-case analysis and intuitively it may not be very efficient in practice, since it uses a static threshold rule, i.e., a threshold that does not adapt throughout the observation of the outcomes. For this reason, we provide further *adaptive* algorithms that are likewise theoretically optimal, and use learning-based methods so as to adapt their threshold according to the observed values. We evaluate these algorithms experimentally, using data from housing prices in France to simulate the prophet setting, and show that they perform much better than the worst-case, theoretic bound.

This study shows that the scenario-based framework can be successfully applied to one of the fundamental online decision-making problems, and complements the work on tree structures. The techniques we develop target adaptive strategies, that do not require full knowledge of the underlying input distribution.

1.2.3 Decision-Theoretic Approaches in Learning-Augmented Algorithms

In this part of the thesis, we study online algorithms that incorporate predictions on the input. The key difference to the robust setting is that we no longer compare performance across input scenarios, but instead we seek to understand how the accuracy of the prediction influences the algorithm's behavior.

A common goal in the learning-augmented framework is to design algorithms

that balance two properties: the *consistency*, which refers to the performance when the prediction is perfect, and the *robustness*, which measures the performance when the prediction is adversarially inaccurate. Prior work in this field often aims to find the best possible trade-off between these two extreme measures. However, this leaves open the question of how to choose among multiple incomparable algorithms that all achieve this optimal trade-off, or more generally, how to choose a good algorithm by taking into account the performance across the entire spectrum of the prediction error.

We address these questions by introducing a framework inspired by some known decision-theoretic approaches in optimization. Specifically, we leverage tools from decision theory that help us differentiate among widely varying algorithms, in terms of their overall performance.

On the deterministic side, we apply *distance-based* metrics that quantify how far an algorithm's performance is from an *ideal* algorithm. Here, we define the concept of an ideal algorithm as a powerful benchmark which is, informally, the best-possible prediction-based algorithm one could ever hope for. On the stochastic side, we draw inspiration from *risk-based* approaches in financial decision-making. Here, the prediction is stochastic, i.e., it is given to the algorithm by a distributional oracle. The goal is to find algorithms that optimize the performance of the algorithm, given a bound on the risk that the algorithm is willing to tolerate.

Together, these approaches enable us to compare algorithms across the entire range of prediction errors and select a suitable one from a class of potential candidates. We apply our framework to three representative problems in online decision-making: ski rental, 1-max search, and contract scheduling. These problems serve as concrete test cases to demonstrate how decision-theoretic tools can improve the design and evaluation of algorithms that rely on predictions.

1.3 Outline of the Thesis

The thesis is structured into two thematic parts. The first part, covered in Chapter 2, provides the theoretical background, context, and state of the art. Specifically, it introduces the framework of online computation competitive analysis as a standard evaluation technique. In addition, it discusses pertinent results from the literature on learning-augmented computation and robust optimization under scenarios.

The second part of the thesis presents, in detail, the original contributions of this thesis. Chapter 3 develops a scenario-based robust optimization framework for tree structures, and discusses connections with fairness.

Chapter 4 extends the traditional formulation of the prophet inequality problem

to a scenario-based setting. In this model, the true environment is unknown but assumed to belong to a known set of possibilities. We propose stopping rules that adapt to this uncertainty and analyze their performance both analytically and experimentally.

Chapter 5 introduces decision-theoretic tools for evaluating algorithms with machine-learned predictions. These approaches allow us to quantify algorithmic performance across the full spectrum of the prediction error, and thus choose the best-possible algorithm within an entire class of otherwise incomparable ones.

Chapter 6 concludes the thesis by summarizing the main findings and highlighting the conceptual connections between the different models and the techniques explored throughout the work. It also outlines several open questions and possible directions for future research, including extensions to more general tree structures such as B-trees and quad-trees, the integration of scenario-based uncertainty with predictions, adaptive scenario discovery, the use of regret as a performance metric, and experimental validation on real-world data.

Part I

Background

In this chapter, we provide the necessary theoretical background for the work that follows. We introduce some core concepts and models from online computation and data structures, with an emphasis on algorithmic decision-making under uncertainty.

This chapter is structured in three parts. Section 2.1 focuses on online algorithms, where the input is revealed over time, and decisions must be made without knowledge of future events. We formally define the model, introduce competitive analysis as a key evaluation technique, and present a selection of representative problems, such as one-max search, prophet inequalities, and contract scheduling. These problems will serve as benchmarks for more nuanced models studied later in the thesis. This section also introduces the learning-augmented setting, where algorithms have access to predictions regarding the input, and provides the basic definitions used to evaluate their performance.

Section 2.2 is devoted to robust optimization, with a particular focus on scenario-based models. It introduces the general framework, and motivation for robust approaches in the context of algorithm design.

Section 2.3 explores two important tree-based data structures: Binary Search Trees (BSTs) and Huffman Trees (HTs). It explains how optimal structures can be constructed when input frequencies are known, and then introduces scenario-based formulations that account for input uncertainty. These formulations motivate the robust optimization methods developed in subsequent chapters.

2.1 Online Algorithms

2.1.1 Competitive Analysis

The online model captures settings in which the input is revealed gradually over time, and decisions must be made step by step without access to future information. An algorithm in this model receives the input as a sequence $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_t)$ and must make irrevocable decisions at each step before observing the next part of the input. This framework is useful in many real-world problems where data arrives sequentially and immediate actions are required. Examples include job scheduling in operating systems where tasks arrive over time, caching and paging

in memory management and online routing in networks when requests come in one after another.

To evaluate the performance of an algorithm under this model, the predominant approach is *competitive analysis*. This technique compares the cost of the online algorithm to the cost of an optimal offline algorithm that knows the entire input sequence in advance. The main idea is to measure how well the online algorithm performs in the worst case, relative to the best possible offline solution.

The concept of competitive ratio, first introduced in Definition 1.1, formalizes this comparison. Let $\text{ALG}(\sigma)$ be the cost incurred by the online algorithm on input σ , and let $\text{OPT}(\sigma)$ be the cost of the optimal offline algorithm on the same input. The algorithm is said to be α -competitive if, for every input sequence σ , it holds that

$$\text{ALG}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma).$$

The smallest such α is called the *competitive ratio* of the algorithm. A relaxed version of this guarantee allows for an additive constant $c \geq 0$, in which case the algorithm satisfies

$$\text{ALG}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma) + c.$$

In most applications of competitive analysis, one assumes that the input sequence is selected by an adversary who aims to maximize the ratio $\text{ALG}(\sigma)/\text{OPT}(\sigma)$, subject to the constraints of the problem. This adversarial view ensures that the algorithm maintains reliable performance even on the worst-possible inputs.

Competitive analysis was first introduced by Sleator and Tarjan [ST85b] in the context of the list update and paging problems, and has since become a fundamental technique for evaluating algorithms in settings with uncertainty. Over the years, it has been applied to a wide range of problems, including the k -server problem [MMS88], metrical task systems [BLS92], rent-or-buy problems [Kar+88]. It has also played a key role in the analysis of graph problems such as online matching [KVV90] and the online Steiner tree problem [IW91], where decisions must be made incrementally as the input is revealed. Beyond these examples, competitive analysis has been used in many other domains such as robotics [ST19] and cloud computing [Lin+13], and has become the standard technique for understanding algorithmic performance under incomplete information.

Example: The Ski Rental Problem

In this problem, a vacationer is on a ski trip of unknown length. Each day, the vacationer must choose whether to rent skis, at a cost of one unit, or to purchase them for a fixed cost $B > 1$. After the skis are purchased, no further costs are

incurred. The challenge lies in the uncertainty: the total number of skiing days is not known in advance, and the vacationer learns each day whether the trip will continue or not. The ski rental problem is one of the most studied examples in the broader class of rent-or-buy problems. Many generalizations have been proposed and applied in different areas, including dynamic TCP acknowledgment [KKR03], the parking permit problem [Mey05], and snoopy caching [Kar+88].

Let us formalize this setting. A request sequence $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_j)$ corresponds to a ski trip that lasts for j days, where each $\sigma_i = Y$ indicates the trip continues. The trip ends after day j , that is, $\sigma_{j+1} = N$ is revealed after the final day. The offline optimum for σ is:

$$\text{OPT}(\sigma) = \min\{j, B\},$$

since one can either rent each day or buy on the first day, depending on which is cheaper.

A deterministic online algorithm must choose on which day i to buy the skis, if any. Let ALG_i denote the algorithm that rents until day $i - 1$ and buys on day i , if the trip is still ongoing. Then the total cost of ALG_i on instance σ is given by:

$$\text{ALG}_i(\sigma) = \begin{cases} j, & \text{if } j < i \\ (i - 1) + B, & \text{if } j \geq i \end{cases}$$

We now determine the best competitive ratio that can be achieved by any deterministic strategy.

► **Theorem 2.1 ([Kar+88]).** The algorithm ALG_B is strictly $(2 - \frac{1}{B})$ -competitive, and this is the best possible competitive ratio for any deterministic algorithm in the ski rental problem. ◀

Proof. We analyze the performance of the algorithm ALG_B , which rents skis for $B - 1$ days and buys them on day B if the trip continues. On any instance σ where the trip lasts j days, the cost incurred by the algorithm is:

$$\text{ALG}_B(\sigma) = \begin{cases} j, & \text{if } j < B, \\ 2B - 1, & \text{if } j \geq B. \end{cases}$$

The offline optimum is given by $\text{OPT}(\sigma) = \min\{j, B\}$. Therefore, the competitive ratio is:

$$\sup_j \frac{\text{ALG}_B(\sigma)}{\text{OPT}(\sigma)} = \max \left\{ \max_{j < B} \frac{j}{j}, \max_{j \geq B} \frac{2B - 1}{B} \right\} = \max \left\{ 1, 2 - \frac{1}{B} \right\} = 2 - \frac{1}{B}.$$

To show that this is optimal, let ALG_i be any deterministic algorithm that buys skis on day i . Consider the instance σ where the trip lasts exactly

$$j = \left\lfloor \frac{(i-1) + B}{2 - \frac{1}{B}} \right\rfloor.$$

Then $\text{ALG}_i(\sigma) = (i-1) + B$, and $\text{OPT}(\sigma) = \min\{j, B\} \leq j$, so:

$$\frac{\text{ALG}_i(\sigma)}{\text{OPT}(\sigma)} \geq \frac{(i-1) + B}{j} \geq 2 - \frac{1}{B}.$$

This shows that no deterministic algorithm can achieve a competitive ratio strictly better than $2 - \frac{1}{B}$. ■

This example serves as an illustrative case for understanding the core ideas behind competitive analysis. While the ski rental problem is seemingly simple, it captures the central difficulty of online computation: making irrevocable decisions without access to future information, and evaluating performance against an ideal offline benchmark.

The ski rental problem can also be studied in a continuous-time setting, where time is treated as a real-valued quantity and the decision to buy skis can be made at any fractional point in time, rather than only at the start of a day. In both discrete and continuous settings, one can also consider randomized algorithms, which assign probabilities to different purchase times instead of choosing a fixed one. Such strategies are less predictable and therefore more resilient to worst-case inputs. A result by Karlin et al. [Kar+90] shows that the optimal randomized algorithm for the ski rental problem achieves a competitive ratio of $\frac{e}{e-1} \approx 1.58$, outperforming the best possible deterministic bound of $2 - \frac{1}{B}$. In this thesis, we focus on deterministic algorithms, as they are better suited for analyzing the objectives we formulate.

2.1.2 Learning-Augmented Algorithms

The study of learning-augmented algorithms focuses on improving the performance by allowing the algorithm access to some prediction about parts of the input that are not yet known. These predictions usually come from machine learning models that are trained on past data and can thus provide useful, though not always accurate, information about the future. The main goal is to design algorithms that make good use of accurate predictions but still perform reliably when the predictions are wrong.

Learning-augmented online algorithms have attracted significant attention in recent years, with many results on a variety of problems, including rent-or-buy models [GP19], graph problems [APT22], secretary problems [Ant+23], packing, covering [BMS20], and scheduling [Lat+20]. The learning-augmented framework was introduced in [KPS24; LV21], and a regularly updated list of works in this area is available in the online repository [LM25].

One of the main challenges in this setting lies in the theoretical analysis of algorithms that rely on predictions. In contrast to standard online algorithms, learning-augmented algorithms must be evaluated along several different criteria, as was shown in the seminal works [LV21] and [KPS24]. These works introduced the following three key notions that guide the evaluation of learning-augmented algorithms.

- *Consistency*: the algorithm performs well when the prediction is perfect.
- *Robustness*: the algorithm does not degrade significantly even when the prediction is arbitrarily wrong.
- *Smoothness*: the performance of the algorithm degrades gracefully as the prediction error increases.

Balancing these objectives is subtle, as they are often in tension with one another. Designing algorithms that achieve favorable trade-offs between them is a central focus in this area.

We now proceed with a more formal discussion of learning-augmented algorithms, along with some preliminary definitions and evaluation metrics that will be used throughout this thesis. We will focus on problems with *single-valued* predictions. Even though single-valued predictions provide very limited information about the input, they still remain important in both theory and practice. Many real-world systems generate such predictions, such as expected demand, duration, or cost. Examples include job runtime prediction in scheduling [KPS24], prediction of the next request time for cached items in caching [LV21], and price prediction in online trading [Sun+21]. These predictions are easy to interpret and integrate into decision-making, making them a natural choice in many applications. At the same time, they offer a clean starting point for understanding how prediction quality affects algorithmic performance.

Let σ denote the input sequence, and let $x_\sigma \in \mathbb{R}$ represent a significant quantity associated with σ . We denote by $y \in \mathbb{R}$ the (possibly inaccurate) prediction of x_σ . For instance, in the ski rental problem, x_σ corresponds to the total number of days the trip lasts. That is, x_σ is the number of skiing days. The quality of the prediction

is measured by the absolute error $\eta := |x_\sigma - y|$. When the context is clear, we will write x instead of x_σ for simplicity.

Assume a *cost* minimization problem. Let A be an online algorithm that receives as input the sequence σ and a prediction y . We denote by $A(\sigma, y)$ the cost incurred by A on input σ using prediction y . The *performance ratio* of A on σ with prediction y is defined as:

$$\text{perf}(A, \sigma, y) = \frac{A(\sigma, y)}{\text{OPT}(\sigma)},$$

where $\text{OPT}(\sigma)$ denotes the minimum cost achievable on σ by an optimal offline algorithm. For *profit* maximization problems, the ratio is defined as $\text{OPT}(\sigma)/A(\sigma, y)$, and the analysis proceeds in a similar way.

The performance of learning-augmented algorithms depends not only on the input but also on the quality of the prediction. To capture this, we evaluate algorithms using two key measures: consistency and robustness.

► **Definition 2.2 (Consistency).** The *consistency* of an algorithm A is defined as the worst-case performance ratio when the prediction is perfect:

$$\text{cons}(A) = \sup_{\sigma} \text{perf}(A, \sigma, x_{\sigma}).$$



Thus, the consistency reflects how well the algorithm performs when the prediction is perfect. In this ideal case, we would like the algorithm to match, ideally, or closely approach the performance of the offline optimum. We say that A is c -consistent if $\text{cons}(A) \leq c$.

► **Definition 2.3 (Robustness).** The *robustness* of an algorithm A is defined as the worst-case performance ratio over all predictions:

$$\text{rob}(A) = \sup_{\sigma} \sup_{y \in \mathbb{R}} \text{perf}(A, \sigma, y).$$

We say that A is r -robust if $\text{rob}(A) \leq r$.



Thus, the robustness measures how well an algorithm performs when the prediction is not only inaccurate, but adversarial. In practice, predictions can be far from correct, and the algorithm must still handle these cases without large losses. A robust algorithm stays within a reasonable performance range, even when the prediction is very poor, and avoids situations where the outcome becomes unacceptably bad.

These two measures are clearly in a tradeoff relation. An algorithm that fully trusts the prediction can perform very well when the prediction is correct but may do poorly when it is not. In contrast, an algorithm that prepares for the worst may miss the chance to benefit from a good prediction. The challenge is to design algorithms that make good use of accurate predictions but still remain reliable when the prediction is wrong. The following definition is one possible formulation of the tradeoff.

► **Definition 2.4.** A learning-augmented algorithm A is *Pareto-optimal* if, for any robustness bound $r > 1$, A is r -robust and achieves the smallest possible consistency among all algorithms satisfying this robustness constraint. ◀

In addition to these two extreme metrics, a broader objective is to express the performance of the algorithm as a function of the prediction *error*. This is formalized in the concept of smoothness, defined below:

► **Definition 2.5 (Smoothness).** Recall that the prediction error is defined as $\eta := |x_\sigma - y|$. Let $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{> 0}$ be a non-decreasing function f , then A is f -smooth if for all input sequences σ and all predictions $y \in \mathbb{R}$,

$$\text{perf}(A, \sigma, y) \leq f(|x_\sigma - y|).$$

◀

Hence, smoothness describes how the performance of an algorithm changes depending on the accuracy of the prediction. While the consistency and the robustness focus on extreme-case performance in regards to the prediction, the smoothness gives a more detailed view by quantifying the performance as a function of the prediction error.

The function f provides an upper bound on the performance ratio based on the prediction error. Ideally, this function should be slowly increasing, in that the performance does not degrade abruptly with the prediction error.

► **Example 2.6 (A Pareto-Optimal Algorithm for Ski Rental).** We now revisit the ski rental problem introduced in Section 2.1.1 to illustrate these definitions. The following deterministic algorithm uses the predicted number of ski days y , the fixed buying cost b , and a parameter $r > 2$ to choose the day on which to buy.

This algorithm was introduced in [KPS24], where it was shown to be r -robust, $\frac{r}{r-1}$ -consistent, and f -smooth, where $f = \frac{r-1}{r-2}$. As shown in that work, it is also Pareto-optimal in the sense of Definition 2.4. That is, the parameter r gives rise to a full family of algorithms, each achieving a different trade-off between consistency

Algorithm 1: Deterministic Learning-Augmented Ski Rental Algorithm

Input: Prediction y , buying cost b , parameter $r \geq 2$

```

1 if  $y \geq b$  then
2    $\lfloor$  Buy on the start of day  $\lceil b/(r-1) \rceil$ ;
3 else
4    $\lfloor$  Buy on the start of day  $\lceil b \cdot (r-1) \rceil$ ;

```

and robustness. This result shows that a simple rule based on a single-valued prediction can lead not only to strong guarantees but also to a complete description of the optimal trade-offs in this setting. ◀

It is perhaps unsurprising that not all of the aforementioned objectives, can be simultaneously optimized. These limitations in analysis carry important implications for the design of learning-augmented algorithms. More concretely, one may choose to focus purely on Pareto-optimality and ignore smoothness, e.g., [CSW23; Lee+24; Sun+21; WZ20]. A different approach is to design algorithms that emphasize smoothness, often without explicitly quantifying the associated trade-offs in consistency or robustness, as in [AKZ22; Ant+23].

Each of these methodologies has its advantages, but they also exhibit certain limitations. For example, Pareto-optimal algorithms can be *brittle*, meaning their performance may drop sharply even when the prediction error is very small [Ele+24]. As a result, they may perform poorly in practical cases where predictions are not exact. On the other hand, approaches that aim for smooth performance often assume a known upper bound on the prediction error, which represents the expected accuracy of the prediction. Algorithm design and analysis are then conducted with respect to this bound [AKZ22; Ant+23]. However, such algorithms may perform poorly when the actual prediction is highly accurate, since they are not designed to take advantage of cases with small prediction error.

2.1.3 Further problems and applications

In this thesis, we study three additional core problems that serve as representative examples for the application of learning-augmented techniques and robust analysis: 1-max search, prophet inequalities, and contract scheduling. Each of these problems presents different algorithmic challenges and provides an opportunity to explore the benefits and limitations of using predictions under uncertainty.

1-Max Search

The 1-max search problem is a central example in the study of online decision-making. It captures a simple setting of selling an indivisible asset, where the objective is to maximize the profit of the sale.

Problem Definition [El-96]. In 1-max search, the input is a sequence σ of prices in $[1, M]$, where M is an upper bound that is known to the algorithm. We denote by x_σ the maximum price in σ , or simply x when the sequence is clear from context.

Upon observing a price, an online algorithm A must either accept the price, in which case it terminates with a profit equal to the said price, or reject, in which case it moves on to the next price in the sequence. We assume that the algorithm is notified when the last price appears, hence its profit is always at least one. The competitive ratio of A is defined as the worst-case ratio between x_σ and $A(\sigma)$ over all sequences σ .

Any algorithm for the 1-max search problem can be viewed as a *threshold algorithm*, which selects a threshold value $T \in [1, M]$ and accepts the first price in the input sequence σ that is at least T . We denote by A_T the threshold algorithm with threshold T .

This problem was introduced in [El-96], where it was shown that choosing $T = \sqrt{M}$ yields the optimal deterministic competitive ratio equal to $\sqrt{M}$. The assumption of knowing M is necessary in order to achieve a bounded competitive ratio.

► **Example 2.7.** To illustrate the 1-max search problem, consider the example shown in Figure 2.1. A sequence of prices arrives one by one, and the algorithm sets a fixed threshold T . In this example, the threshold is set to $T = 25$, and the algorithm accepts the first price that reaches or exceeds this value. At time step 5, the price is 30, which is above the threshold, so the algorithm selects this value and terminates. However, the maximum price in the sequence, $x_\sigma = 50$, appears later at time step 7. This example highlights the main difficulty of the problem: the algorithm must make a decision without knowing what values will come next. It also shows how the performance depends on the relationship between the threshold and the actual position of the maximum value. ◀

Another problem related to 1-max search is the *one-way trading* problem. In this setting, a trader starts with an initial budget, typically normalized to 1, and observes a sequence of exchange rates, each within the interval $[1, M]$. Unlike 1-max search, where the asset is indivisible, here the trader decides what fraction of the remaining budget to exchange at each step. The total number of exchange rates

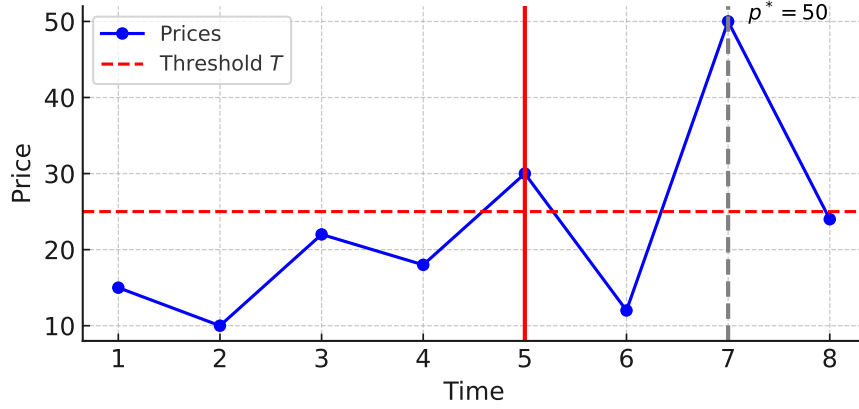


Figure 2.1: Threshold algorithm applied to a sequence of prices.

is unknown in advance, but the trader is notified when the final rate is reached and must exchange all remaining budget at that point. One-way trading can be seen as a randomized version of 1-max search and has competitive ratio $\mathcal{O}(\log M)$ [El+01].

1-max search and one-way trading play an important role in online decision-making and financial algorithms. As discussed in the survey [MAS14], they have motivated a broad range of variants and extensions, all studied through the lens of competitive analysis.

On the other hand, a different line of work focuses on Bayesian methods [RS81], where decisions are based on known probability distributions. These methods assume that the input follows a predictable pattern, in contrast to the worst-case guarantees provided by competitive analysis. The competitive approach aims to ensure stable algorithmic performance even when no information is available about the input distribution.

One can define a natural learning-augmented version of the 1-max search problem, in which the algorithm is given a prediction y of the maximum price x_σ in the input sequence. This model was explored in [Sun+21], where the authors introduced a class of Pareto-optimal algorithms, and in [AKZ22], which gave algorithms with smoothness guarantees. However, it was later shown in [BP25; Ele+24] that such algorithms may suffer from brittleness: even small prediction errors can result in a significant loss in performance. An alternative algorithm was proposed in [AKZ22], which introduces a confidence parameter δ that bounds the possible prediction error. This algorithm achieves smoother performance as a function of the error, but it does not perform well when the prediction is very accurate.

Prophet Inequalities

Prophet inequalities are a central model in the study of online decision-making under uncertainty. These problems involve algorithms that must sequentially decide whether to accept the current offering or continue in the hope of a better future value, as a sequence of independent random variables is revealed one by one. The efficiency of these algorithms is evaluated against a theoretical "prophet" who, knowing the entire sequence in advance, can always select the optimal value.

Problem Definition [PF66]. Suppose there are n boxes, each containing a value $X_i \geq 0$, drawn independently from known distributions $D_1, \dots, D_n$. As the boxes are opened one by one, the algorithm (e.g., "gambler") observes the value of each box and must decide whether to accept it and stop, or reject it and proceed to the next box. The selected value is the algorithm's payoff. If no value is selected, the payoff is defined to be zero. The performance of the algorithm is measured against the expected maximum value, namely $\mathbb{E}[\max_{i \in [n]} X_i]$, which an omniscient "prophet" could obtain by selecting the best value in hindsight.

This problem was introduced by Gilbert and Mosteller in 1966 [PF66] and further studied in [KS77; KS78], which showed that an optimal strategy guarantees, in expectation, at least half of the prophet's benchmark, and that this bound is tight.

The competitive ratio measures how much better the prophet can perform compared to the algorithm. For a given input sequence of distributions $\sigma = (D_1, \dots, D_n)$, let $\mathbb{E}[\text{ALG}(\sigma)]$ be the expected reward obtained by the algorithm, and let $\mathbb{E}[\max_{i \in [n]} X_i]$ be the expected reward of the prophet. The competitive ratio of the algorithm is defined as

$$\text{CR}(\text{ALG}) = \sup_{\sigma} \frac{\mathbb{E}[\max_{i \in [n]} X_i]}{\mathbb{E}[\text{ALG}(\sigma)]}.$$

An algorithm is said to be α -competitive if this ratio is at most α .

The seminal work [Sam84b] introduces a simple but efficient single *threshold* strategy. The strategy chooses a threshold τ satisfying

$$\mathbb{P}\left[\max_{i \in [n]} X_i > \tau\right] = \frac{1}{2},$$

and accepts a value X_i if and only if it is at least τ . While this strategy does not guarantee the best outcome for each individual sequence, it consistently achieves a competitive ratio equal to 2 when compared to the prophet's benchmark.

► **Example 2.8 ([KS78]).** Consider two boxes. The first box always contains the value $X_1 = 1$. The second box contains $X_2 = \frac{1}{\epsilon}$ with probability ϵ , and 0 otherwise. The expected value of the maximum is

$$\mathbb{E}[\max\{X_1, X_2\}] = \epsilon \cdot \frac{1}{\epsilon} + (1 - \epsilon) \cdot 1 = 2 - \epsilon.$$

Since the algorithm must decide whether to accept the first value without knowing the second, the best it can do is take $X_1 = 1$. Any algorithm has expected reward at most 1, while the prophet, who sees both values in advance, selects the maximum and achieves $2 - \epsilon$ in expectation. This implies that the competitive ratio of any algorithm is at least $2 - \epsilon$, confirming that the bound of 2 is tight. ◀

We note that the prophet inequality problem has been studied predominantly under the assumption that all distributions are known in advance. However, this requirement often does not hold in real-world scenarios where the decision-maker may not have a precise knowledge of the distributions.

Contract Scheduling

Contract scheduling is a central problem in the design of real-time systems and in bounded-resource reasoning [BFZ03; LAH14; RZ91]. It also relates to well-known models such as online bidding [Ana+22] and searching on the infinite line [BD70].

Problem Definition [RZ91]. A contract algorithm is an algorithm that improves its solution quality with more time, but must run for a fixed period before producing a result. It cannot be stopped at an arbitrary point and expected to return a meaningful output. In this setting, a schedule X is a non-decreasing sequence $x_1, x_2, \dots$, where x_i is the amount of time allocated to the i -th execution of the contract algorithm. The goal is to design a schedule such that at any possible interruption time T , at least one contract has finished and its result can be returned.

Let $\ell(X, T)$ denote the length of the longest contract that is completed by time T . The performance of the schedule is measured by the *acceleration ratio* [RZ91]:

$$\text{acc}(X) = \sup_T \frac{T}{\ell(X, T)}.$$

► **Example 2.9.** Consider the schedule $X = (1, 2, 4, 8, \dots)$. If the interruption occurs at time $T = 14$, the algorithm will have completed contracts of lengths 1, 2, and 4, while the contract of length 8 is only partially done. In this case, $\ell(X, 14) = 4$, and $14/\ell(X, 14) = 3.5$. ◀

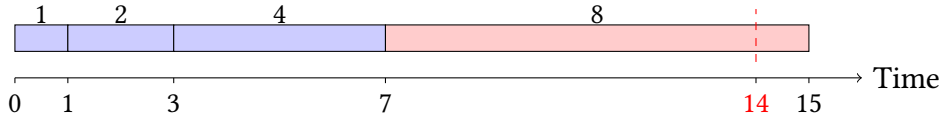


Figure 2.2: Execution of the schedule $X = (1, 2, 4, 8, \dots)$ with interruption at time $T = 14$.

Figure 2.2 illustrates the schedule described in Example 2.9. The blue segments show the contracts that have been completed at time $T = 14$, and the red segment indicates a contract that was still running when the interruption occurred at time $T = 14$.

In this setting, the best possible acceleration ratio is known to be 4 [RZ91], and is achieved by the doubling schedule of Example 2.9. This problem has been extensively studied in the context of real-time systems and AI planning under bounded resources [BD94; ZR96]. More recent works have studied versions of the problem where the algorithm (i.e. the scheduler) receives a prediction y of the interruption time T , and the prediction error is defined as $\eta = |T - y|$ [AK23a].

► **Definition 2.10.** Assuming a prediction y on the interruption time, the consistency of a schedule X is defined as

$$\text{cons}(X) = \frac{y}{\ell(X, y)},$$

whereas the robustness is defined as

$$\text{rob}(X) = \sup_{T \geq 1} \frac{T}{\ell(X, T)}.$$



Note that the definition of consistency captures the efficiency of the schedule, i.e., it is related to the largest contract completed by the predicted time, whereas the robustness is the same as the acceleration ratio of the schedule.

It was shown in [AK23a] that the best-possible consistency of any 4-robust schedule is equal to 2. However, as in the case of 1-max search, these schedules can be brittle [Ele+24]. That is, even a very small error in the prediction can lead to a significant drop in the performance. More precisely, for any $\epsilon > 0$, there exists a predicted time y and actual time T such that $|T - y| = \epsilon$, and any schedule that is both 4-robust and 2-consistent satisfies

$$\ell(X, T) \leq \frac{T + \epsilon}{4}.$$

2.2 Robust Optimization

In many optimization problems, the input data is uncertain or only partially known at the time a decision must be made. This uncertainty can significantly affect the feasibility and quality of the resulting solutions. Robust Optimization (RO) is a framework that addresses this issue by searching for solutions that remain feasible and efficient under a range of possible data realizations. The main goal is to safeguard the decision-maker against the worst possible outcomes within a given uncertainty set.

Robust optimization differs from stochastic optimization, which models uncertainty using probability distributions. In stochastic optimization, the goal is to typically satisfy constraints with high probability, assuming that the data follow a known or partially known distribution [Spa03]. In contrast, robust optimization avoids probabilistic assumptions and requires solutions to remain feasible for all values within a specified *uncertainty set* [BGN09]. While the stochastic approach can be less conservative, it relies on accurate knowledge of the underlying distribution, which may not be available or meaningful in many practical settings.

The origins of robust optimization go back to the 1950s, when modern decision theory started to develop. A central concept that was introduced at that time was Wald's maximin model, in which the decision is chosen so that it leads to the best-possible outcome among all scenarios, and specifically for the worst-case scenario [Wal45; Wal50]. The idea is closely related to zero-sum game theory, where one player (the decision maker) chooses an action first, and nature or uncertainty responds with the worst possible scenario. Over time, this idea has been formalized into a general framework and applied in various areas, including operations research [BS04], control theory [KPZ90], finance [Fab+07] and more.

To formalize the basic idea, consider a linear program of the form

$$\min_x \{c^T x : Ax \leq b\}.$$

Here, the vector x is the decision variable, and the input data (c, A, b) may be uncertain. In robust optimization, solving this linear program reduces to solving a family of linear programs, one for each possible realization of (c, A, b) within a given uncertainty set $\mathcal{U}$. A solution is considered *robust* if it satisfies all constraints for every instance in $\mathcal{U}$ [BGN09]. This leads to the min-max formulation:

$$\min_{x \in \mathcal{X}} \max_{(c, A, b) \in \mathcal{U}} c^T x \quad \text{subject to} \quad Ax \leq b.$$

This model captures the idea of accounting for the worst case, and is particularly

useful in applications where constraint violations are not acceptable, or where it is difficult to estimate the probability of uncertain events.

While traditional robust approaches focus on the worst-case performance over a continuous uncertainty set, in many practical situations it is useful to model uncertainty using a finite set of discrete possibilities. This leads to a specific method known as *scenario-based robust optimization*, which we discuss in the following section.

2.2.1 Scenario-Based Robust Optimization

One practical approach to robust optimization is *scenario-based optimization*, where uncertainty is modeled through a finite set of possible scenarios. Each scenario represents a different realization of the uncertain data. This approach is especially useful when the uncertainty set is complex or when historical data are available. This framework has been studied in fields such as planning [MVT17], scheduling [SG23], network optimization [KKZ15; KZ15] and many more. Scenario-based robust optimization has been extensively applied to the study of *scheduling* problems under uncertainty. Examples include the minimization of completion time [MMS13], flow-shop scheduling [KKZ12], and just-in-time scheduling [GS21]. We refer to [SG23] for a recent survey of many results related to scenario-based scheduling.

Formally, let $\mathcal{S} = \{\xi^{(1)}, \xi^{(2)}, \dots, \xi^{(N)}\}$ be a finite set of scenarios, where each $\xi^{(i)}$ specifies a possible instance of the problem data. The goal is to find a solution x that performs well across all scenarios, as expressed by

$$\min_x \max_{i \in \{1, \dots, N\}} f(x, \xi^{(i)}) \quad \text{subject to} \quad g_j(x, \xi^{(i)}) \leq 0 \text{ for all } i \text{ and } j.$$

Here, $f(x, \xi^{(i)})$ is the objective function evaluated under scenario $\xi^{(i)}$, and $g_j(x, \xi^{(i)}) \leq 0$ represents the j -th constraint that must be satisfied in that scenario. The functions g_j capture the feasibility requirements of the problem, such as resource limits, logical conditions, or physical laws. This min-max formulation seeks to optimize the performance of x in the worst-case scenario among those in $\mathcal{S}$. In some applications, a similar model appears with a max-min objective, depending on whether the focus is on maximizing gains or minimizing losses. This formulation follows the scenario-based robust optimization framework developed in [BGN09; CG08].

► **Example 2.11 (Investment under Uncertainty).** To illustrate, consider an investment problem where the return $R_\delta(x)$ depends on a vector of investment decisions x and on the market condition δ at the end of the investment period.

Suppose we have N possible market scenarios $\delta_1, \dots, \delta_N$, obtained either from a stochastic model or from historical data. The scenario optimization problem is written as:

$$\max_x \min_{i=1, \dots, N} R_{\delta_i}(x).$$

This model corresponds to selecting a portfolio x that maximizes the minimum return over all scenarios. The optimal solution x^* guarantees that the return will be no worse than $R^* = \min_i R_{\delta_i}(x^*)$, and this result is robust with high probability even for future unseen market states [Cal13; PRC12]. ◀

While this worst-case approach helps ensure that a solution performs well even in the most difficult situations, it can sometimes be too pessimistic, which can be considered a disadvantage, e.g., in areas like financial optimization. One way to make the scenario-based approach more flexible is to allow the exclusion of a small number of very unlikely scenarios. Another widely used idea is to rely on alternative risk measures. A common example is *Value at Risk* (*VaR*), which identifies the loss level that will not be exceeded with high probability. Building on this, *Conditional Value at Risk* (*CVaR*) focuses on the expected loss in the worst ε -fraction of cases [RC18; SSU08]. Unlike the worst-case cost, which only considers the single most difficult scenario, CVaR provides a more stable measure by averaging over the most extreme outcomes. We will return to these ideas later in Chapter 5, where we will use them in the evaluation algorithm performance under uncertainty.

2.3 Tree Data Structures

Data structures are a fundamental concept in Computer Science, offering a foundation for efficient data management. Among these, *tree structures* occupy a central role due to their hierarchical organization, which allows for relatively easy, and recursive approaches to a variety of algorithmic problems. A tree is an abstract data type consisting of nodes connected in a parent-child relationship. Each node, has exactly one parent (except for the root), and may have zero or more children. This acyclic and branching structure distinguishes trees from linear data models and makes them especially useful for representing structured information such as file systems, expression trees, and search spaces.

A particularly important family of trees are *binary trees*, where each node has at most two children, typically referred to as the left and right child. When the order of these children matters, the tree is called an *ordered binary tree*, and its structure directly impacts the efficiency of operations. Many classic data structures used in

algorithms are based on binary trees, with Binary Search Trees (BSTs) and Huffman Trees (HTs) being two of the most well-known and widely applied variants.

Binary Search Trees are used to maintain a dynamically changing set of ordered keys while supporting fast search, insertion, and deletion operations. Their performance depends on the shape of the tree, which in turn is influenced by the access frequencies of the keys. Huffman Trees, on the other hand, arise in the context of data compression. They form the basis of Huffman coding [Huf52], a well-known algorithm for constructing prefix-free binary codes that minimize the expected length of encoded messages. This is achieved by assigning shorter codewords to more frequent symbols, making the encoding efficient in terms of space.

In the sections that follow, we explore these two structures in detail. We present the standard formulations of BSTs and HTs, and present some properties that will be useful in our study in Chapter 3.

2.3.1 Binary Search Trees

A BST stores n ordered keys, which we may assume are labeled $\{1, \dots, n\}$ or by symbols such as $A, \dots, Z$. The BST satisfies the *ordering property*: for each internal node, the key in the left subtree is smaller and the key in the right subtree is larger. A search operation starts at the root and follows a path down the tree, moving left or right depending on how the search key compares to the current node.

A BST can also be described by its *level vector* $L \in \{1, \dots, n\}^n$, where L_i is the level (or depth) of the i -th key in the tree. The root has level 1, and the level increases by one along each edge. Given a *frequency vector* F , which represents the access frequency of each key, the expected cost of searching in the tree is defined as:

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

Knuth presented a dynamic-programming algorithm [Knu71], which outputs a tree of optimal cost, given a frequency vector F .

► **Example 2.12 (An optimal BST for seven keys).** Consider a BST built on the keys A through G , with the following access frequencies:

Key	A	B	C	D	E	F	G
Frequency	3	2	6	8	4	2	1

Using Knuth's algorithm, the optimal BST for this distribution is shown in Figure 2.3, with level annotations on the left:

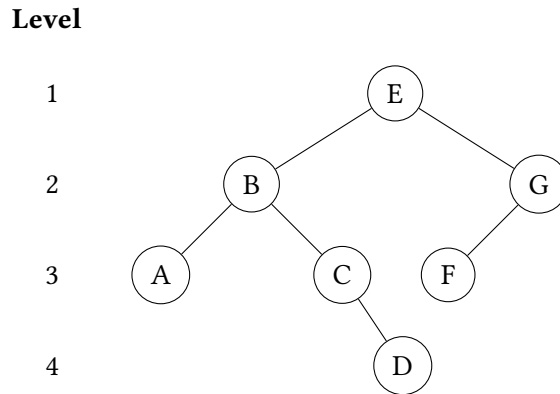


Figure 2.3: An optimal BST for keys A to G with the given frequency distribution.

The level vector for this tree is $L = [3, 2, 3, 4, 1, 3, 2]$, and the normalized frequency vector is:

$$F = \left[\frac{3}{26}, \frac{2}{26}, \frac{6}{26}, \frac{8}{26}, \frac{4}{26}, \frac{2}{26}, \frac{1}{26} \right]$$

Using the formula

$$\text{cost}(L, F) = \sum_{i=1}^7 L_i \cdot F_i,$$

we obtain:

$$\text{cost}(L, F) = \frac{3 \cdot 3 + 2 \cdot 2 + 3 \cdot 6 + 4 \cdot 8 + 1 \cdot 4 + 3 \cdot 2 + 2 \cdot 1}{26} = \frac{60}{26} \approx 2.31$$



While Knuth's algorithm computes the optimal tree in $O(n^2)$ time, this runtime may become prohibitive for large n [Knu71]. To address this, Knuth also proposed two heuristic methods for building nearly-optimal trees efficiently:

- **Rule I (Root-max):** Place the key with the highest frequency at the root, and then apply the same rule recursively to the left and right subtrees.
- **Rule II (Bisection):** Choose the root so that the total weight (frequency) of the left and right subtrees is as balanced as possible, and apply this rule recursively.

These heuristics construct nearly optimal BSTs in $O(n \log n)$ time and $O(n)$ space, making them suitable for large-scale applications. Mehlhorn [Meh75] later provided an even simpler linear-time algorithm that closely approximates the optimal cost. His method uses the bisection rule and leverages the observation that, under balanced splits, the weights along paths tend to decrease geometrically. More precisely, [Meh75] proved that this strategy yields a tree whose weighted path length (from the root to the searched key) is at most

$$2 + \frac{H}{1 - \log(\sqrt{5} - 1)},$$

where H is the entropy of the key distribution. In comparison, the optimal expected search cost cannot be smaller than $\frac{H}{\log 3}$, showing that the approximation is remarkably tight.

2.3.2 Huffman Trees

Huffman Trees are fundamental data structures in lossless data compression, designed to assign binary codewords to symbols such that the total expected length of the encoded message is minimized. Given a set of keys (e.g., characters) and a frequency vector representing how often each key appears, Huffman's algorithm [Huf52] constructs an optimal prefix-free binary tree. The more frequently a key appears, the closer it is placed to the root, ensuring that frequent symbols receive shorter codes.

Formally, the cost of a Huffman Tree is defined as:

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

where F_i is the frequency of key i and L_i is its level (depth) in the tree, with the root at level 0.

The algorithm proceeds by iteratively merging the two least frequent nodes to form a new internal node. This process continues until only one node remains, which becomes the root of the Huffman Tree. Internal nodes store only cumulative frequencies and references to children, while leaf nodes contain the original symbols. As a result, the final tree has at most n leaf nodes and $n - 1$ internal nodes. The canonical labeling assigns 0 to left edges and 1 to right edges. Traversing the tree from the root to any leaf and collecting these labels yields the codeword for the

corresponding symbol. The prefix-free property ensures that no codeword is a prefix of another, allowing for unambiguous decoding.

The standard implementation of Huffman's algorithm uses a priority queue to repeatedly select the two nodes with the lowest frequencies. This yields an overall time complexity of $O(n \log n)$, where n is the number of distinct symbols [Huf52]. However, when the input is sorted, an optimized linear-time algorithm using two queues can be employed. This method guarantees that the next smallest pair is always available at the head of one of the two queues, reducing the computational overhead.

► **Example 2.13 (A Huffman tree for seven letters).** Suppose we want to encode the letters A through G using the following frequency distribution:

Letter	A	B	C	D	E	F	G
Frequency	3	2	6	8	4	2	1

Applying Huffman's algorithm to this distribution yields the binary tree shown in Figure 2.4. Each edge is labeled with a binary digit corresponding to the code prefix, and the level of each node is indicated in the column to the left:

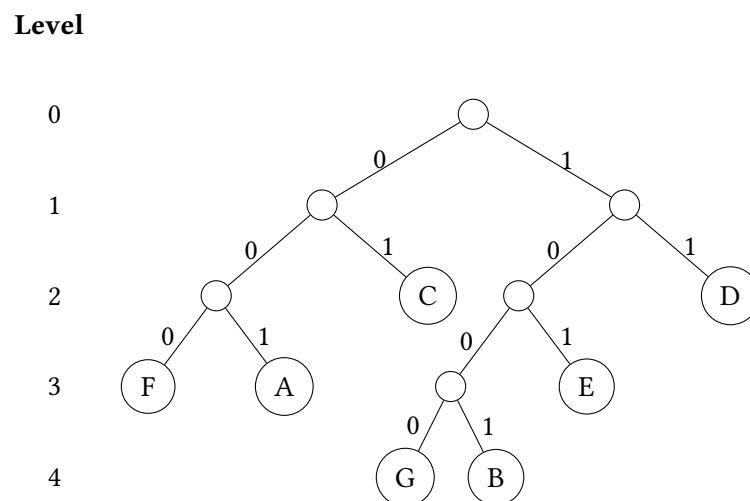


Figure 2.4: A Huffman tree for letters A–G with edge labels indicating prefix-free binary codes and level annotations on the left.

From this structure, we derive the following prefix-free codes:

Letter	Codeword
A	001
B	1001
C	01
D	11
E	101
F	000
G	1000



In this example, the most frequent symbols, such as *D* and *C*, are assigned the shortest codewords, while less frequent symbols, like *G* and *B*, receive longer ones.

Part II

Contributions

3

Scenario-Based Robust Optimization of Tree Structures

3.1 Introduction

Suppose we are tasked with encoding and transmitting text in a given language as efficiently as possible, aiming to minimize the expected number of bits used. When the frequency distribution of the alphabet is known in advance, meaning the language is fixed, this can be done optimally using the well-known technique of Huffman coding [Huf52], as discussed in Section 2.3. But what if the language is not known beforehand, and it is only known that it will be one of a few possibilities, such as English, Italian, or Finnish? In that case, the goal is to design a single code that performs well across all options, and in particular, remains efficient in the worst case.

A similar situation arises in the context of Binary Search Trees (BSTs). Given a set of n keys and a known frequency distribution, the objective is to construct a BST that minimizes the expected number of comparisons during search operations. This problem, and its exact solution using dynamic programming, is covered in Section 2.3. However, in many settings, the frequency distribution may not be fixed. Instead, we may be given k different scenarios, each describing a possible frequency input. The challenge is then to construct a single BST that performs well across all scenarios, including the one that turns out to be the worst for the chosen tree.

Motivated by these examples, this chapter investigates the design of tree structures under uncertainty, modeled by a given set of scenarios. Each scenario corresponds to a frequency vector over the same set of keys. The goal is to construct a single tree, either a BST or a Huffman Tree, that achieves good performance across all scenarios. This type of problem falls under the general framework of scenario-based robust optimization, which is introduced in Section 2.2. To our knowledge, this precise framework has not yet been applied to data structures such as BSTs or to compression methods like Huffman coding.

To evaluate the performance of a solution, we consider two different measures. For BSTs, we use the competitive ratio, defined as the maximum, over all scenarios s , of the ratio between the cost of the constructed tree and the cost of an optimal tree under s . This notion originates in the study of online algorithms and is commonly used in the analysis of self-adjusting data structures, such as in the work on Splay Trees by Sleator and Tarjan [ST85b]. A general overview of competitive analysis is

provided in Section 2.1.1. For Huffman Trees, we use the measure of *regret*, defined as the maximum difference, over all scenarios s , between the cost of the constructed tree and the cost of the optimal tree for that scenario. Regret minimization has been used in areas such as database query processing [Nan+10; XWL20], and is known to provide stricter worst-case guarantees (see Corollary 3.12).

The remainder of this chapter develops formal models for these problems and presents algorithms for constructing trees that are efficient with respect to both evaluation criteria.

3.1.1 Contribution

We begin with the study of robust BSTs in Section 3.2. 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 3.4). We also give an algorithm that constructs a BST of competitive ratio at most $\lceil \log_2(k + 1) \rceil$ (Theorem 3.5), 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 3.7).

In Section 3.3, 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 3.8). 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 3.11).

In Section 3.4, 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 3.5, we provide an experimental evaluation of all our algorithms over real data. To be able to evaluate and compare the algorithms from Section 3.2 and Section 3.3, we also provide mixed integer linear program formulations for all objectives, which allows us to compute the optimal trees for small instances.

In Section 3.6, we explore how the *entropy* of a distribution relates to the structure and cost of BSTs. We build on earlier results, in particular Mehlhorn's construction [Meh75], and use regret as the comparison measure. We show that a tree built

for the average of two scenarios has regret at most $O(\log \log n)$ compared to the best tree for each scenario. We also extend this idea to k scenarios and show that the regret remains bounded by $O(\log k) + O(\log \log n)$.

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 3.4), whereas for HTs we use a non-trivial reduction from the EQUAL-CARDINALITY PARTITION problem (Theorem 3.8). 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.

3.1.2 Further Related Work

This section discusses related research beyond the material already covered in Part I, with a focus on recent developments that are closely related to the scenario-based approach considered in this chapter.

A growing body of work in Machine Learning has explored how to enhance data structures using predictions. These learning-augmented approaches assume access to some information about future queries or data distributions. While learning-augmented data structures were not covered in the background part of this thesis, the general concept of using predictions was introduced in the context of online algorithms in Section 2.1.2. Examples of learning-augmented data structures include skip lists [FSZ24; ZKH24], binary search trees [LLW22; Xin+22], B-trees [Kra+18], and rank/select dictionaries [BFV22]. A broader survey can be found in [FV19]. These works aim to design structures that perform well when predictions are accurate, and degrade gracefully when they are not. In the scenario-based setting considered in this chapter, the k scenarios can be viewed as k different predictions, and the goal becomes to construct a single structure that performs well even under the worst-case prediction.

There is also a body of work analyzing BST behavior under various probabilistic or adversarial conditions. It is known, for example, that when frequencies are chosen uniformly at random, the optimal BST tends to resemble a complete tree and has logarithmic height with high probability. At the other extreme, there are frequency profiles for which the optimal BST has linear height. Intermediate

regimes have been studied using small random perturbations to the frequency vector [Fis81; MR07]. Other works have explored bicriteria optimization for BSTs, such as minimizing both the average and maximum weighted depths [MM20].

Another direction explores binary search with distributional predictions, aiming to minimize the expected query complexity. One approach builds on the well-known algorithm of [Meh75] to incorporate predictions about the key sought, measuring error via the earth mover distance η between the true and the predicted distributions. The resulting query complexity is bounded by $O(H(p) + \log(\eta))$, where $H(p)$ is the entropy of the true distribution, and this bound is tight, as shown in [Din+24]. In contrast, we study optimal binary search trees via dynamic programming [Knu71], assuming k possible scenarios with one being correct. We focus on obtaining exact, non-asymptotic performance guarantees, and do not model prediction error explicitly.

One of the earliest work on binary search trees studied the problem of building a *universal tree* T for a given convex and compact set of possible distributions $\mathcal{P}$ [Fis81]. The proposed approach is to identify the distribution $q \in \mathcal{P}$ with the highest entropy $H(q)$, and to build the optimal tree T for q , see Section 3.6 for a short introduction into entropy. The cost of T on any distribution $p \in \mathcal{P}$, is then upper bounded by $H(q) + 2$. The additive constant $+2$ is reduced to $+1$, in the setting considered in this chapter, where we have only distribution on keys, and do not consider queries on elements between the given keys. Note that the upper bound $H(q) + 1$ can be arbitrarily far from the cost of the optimal search tree for distribution p , which is roughly $H(p) + 1$. In this sense, the proposed tree is not fair in terms of regret, and also does not minimize the worst regret of T over all distributions in $\mathcal{P}$. This contrast with the work presented in this chapter, where we address fairness and worst case guarantees, and consider a finite cardinality set of distributions, rather than a convex continuous set.

Last, we discuss issues related to the concept of fairness in algorithm design. Fairness, defined as the equitable treatment of users sharing a common resource, has been studied in settings such as optimal stopping [AK22; BJS09] and knapsack-like allocation problems [Lec+24; PKL21]. However, to our knowledge, no prior work has addressed fairness in the context of data structures from a worst-case, analytical perspective, despite the fact that such questions are naturally motivated. For instance, one may wish to design a data structure, such as a BST, that ensures equitable search times across multiple competing users. The notion of Pareto-dominance as a fairness criterion has recently received attention in machine learning, see [Mar+21; MBS20]. In our work, we do not rely on learning oracles, and instead seek Pareto-optimal solutions that capture group fairness through regret.

3.2 Robust Binary Search Trees

3.2.1 Background and Measures

We refer to Section 2.3.1 for definitions and related results on BSTs. In its standard form, a BST stores n keys from an ordered set; without loss of generality, we assume the keys are indexed by $\{1, \dots, n\}$. 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 subtree and smaller than all keys in its right subtree. A search operation begins at the root and descends through the tree based on key comparisons.

As discussed in Section 2.3.1, a BST can be represented equivalently by a *level vector* $L \in \{1, \dots, n\}^n$, where L_i indicates the level (or depth) of key i , with the root at level 1. The following condition characterizes valid level vectors for BSTs.

► **Definition 3.1.** A vector $L \in \{1, \dots, n\}^n$ is a *level vector* of a BST if the following two conditions hold:

- For every $1 \leq i < j \leq n$ such that $L_i = L_j$, there exists an index $i < r < j$ with $L_r < L_i$.
- The vector L is *member-wise minimal*, i.e., decreasing any coordinate L_i would violate the first condition.



This definition ensures that the level vector corresponds to a valid binary search tree: the first condition enforces the hierarchical structure between keys on the same level, and the second condition excludes vectors that are structurally valid but contain redundant levels.

► **Example 3.2.** Consider the vector $L = (3, 1, 3)$. It satisfies the first condition because the two keys at level 3 are separated by the key at level 1. However, this vector does not correspond to a BST since level 2 is missing. In fact, both 3s can be reduced to 2s without violating the first condition, which means that L is not member-wise minimal. The second condition in Definition 3.1 ensures that such level vectors are excluded.



This representation allows us to express the *average cost* of a BST relative to a given frequency vector F as the inner product

$$\text{cost}(L, F) = \sum_{i=1}^n L_i \cdot F_i. \quad (3.1)$$

Given a frequency vector F , a level vector L minimizing $\text{cost}(L, F)$ can be computed using a dynamic programming algorithm in time $O(n^2)$ [Knu71].

Next, we formally define the *robust BST* problem. The input consists of k frequency vectors $F^1, \dots, F^k$, referred to as *scenarios*. We consider three performance measures for evaluating the quality of a single level vector L , each giving rise to a distinct minimization objective.

- *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.

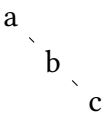
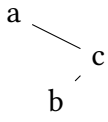
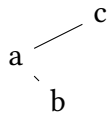
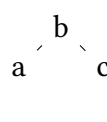
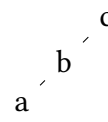
- *Regret*: here, the objective is to minimize the quantity

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

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 3.3.** 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.

					
cost for F^1	11/4	9/4	7/4	3/2	5/4
cost for F^2	17/9	16/9	16/9	17/9	19/9
worst cost	11/4	9/4	16/9	17/9	19/9
competitive 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 3.1, 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 advance, similar to the competitive analysis of online algorithms [BE98]. 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 3.4.

3.2.2 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 3.4.** 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 [GJ79]. An instance of this problem consists of a list $a_1, \dots, a_m$ of non-negative integers, and the goal is to decide whether there exists a binary vector $b \in \{0, 1\}^m$ with $\sum_i b_i a_i = \sum_i (1 - b_i) a_i$. 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

$$F^1 = (a_1, 0, w, a_2, 0, w, a_3, 0, \dots, w, a_m, 0) \text{ and}$$

$$F^2 = (0, a_1, w, 0, a_2, w, 0, a_3, \dots, w, 0, a_m),$$

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 $i = 1, 2, \dots, m$. 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 $i = 1, 2, \dots, m$, 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 3.1 for an illustration.

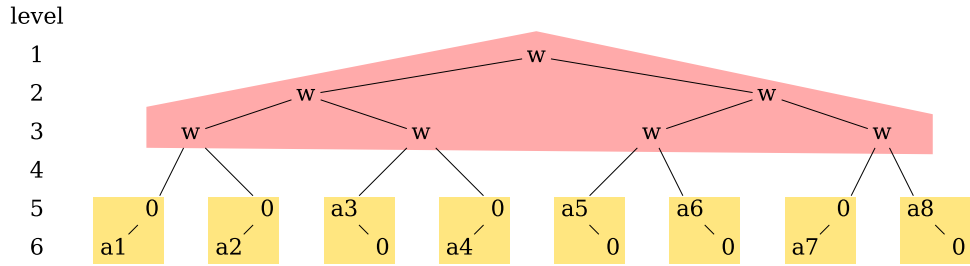


Figure 3.1: The BST corresponding to the binary vector $b = \{11010010\}$ in the NP-hardness proof construction. Nodes are labeled with the frequency of their keys in F^1 .

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 $i = 1, 2, \dots, m$ 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.

Algorithm 2: R-BST**Input:** k scenarios described by frequency vectors $F^1, \dots, F^k$ **Output:** A robust BST represented as a level vector L

```

1 foreach scenario  $s \in \{1, \dots, k\}$  do
2   | Compute the optimal tree  $L^s$ ;
3 Call  $A(1, n, 1)$ 

```

The proof extends to the other two metrics, namely the competitive ratio, and the regret. Observe that level vector L^b for $b = (0, \dots, 0)$ minimizes the cost for the first scenario, while for $b = (1, \dots, 1)$ it minimizes the cost for the second scenario, and that both costs are equal to W . Hence, it is also NP-hard to decide if there is a BST of competitive ratio at most $(W + (\sum a_i)/2)/W$. Similarly, it is NP-hard to decide if there is a BST of regret at most $(\sum a_i)/2$. ■

Next, we present an algorithm that achieves the optimal competitive ratio. Algorithm 2 first computes optimal trees with level vector L^s for each scenario $s = 1, \dots, k$, 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 the key(s) in $[i, j]$ at the minimum level in the optimal trees across all scenarios L^s (lines 4 and 5) 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 6).

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

► **Theorem 3.5.** 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 . ◀

Proof. Let $R = \lceil \log_2(k+1) \rceil$ be the claimed competitive ratio. We will show that in line 5 of Algorithm 3, we have $\ell \leq Rv$. This implies that for every key a , and every scenario s , we have $L_a \leq R \cdot L_a^s$, which suffices to prove the theorem.

First, we observe by the definition of level vectors that for every scenario s , there is at most one key $i \leq r \leq j$ with $L_r^s = v$. This implies that $|S| \leq k$. Furthermore, with each recursive call, the level ℓ increases by 1, while either v increases as well, or $|S|$ decreases to at most $\lceil (|S| - 1)/2 \rceil$. It remains to show that after at most R recursive calls, the value v has strictly increased.

To this end, given some x , define $h(x)$ as the largest value such that $\lceil (h(x) - 1)/2 \rceil = x$. This function is $h : x \mapsto 2x + 1$. The sequence $0, h(0), h(h(0)), \dots$ is thus

Algorithm 3: Recursive procedure $A(i, j, \ell)$

```

1 Function  $A(i, j, \ell)$ :
2   if  $i > j$  then
3     return ;                                // Empty interval ends recursion
4   Let  $v$  be the minimum level  $L_r^s$  over  $i \leq r \leq j$  and  $1 \leq s \leq k$ ;
5   Let  $S$  be the ordered set of keys  $r \in [i, j]$  such that there exists a scenario
       $s$  with  $L_r^s = v$ ;
6   Let  $m \in S$  such that both  $|S \cap [i, m-1]|$  and  $|S \cap [m+1, j]|$  are at most
       $\lceil (|S| - 1)/2 \rceil$ ;
7   Set  $L_m = \ell$ ;
8    $A(i, m-1, \ell+1)$  ;                        // Left recursion
9    $A(m+1, j, \ell+1)$  ;                        // Right recursion

```

a sequence of numbers of the form $2^i - 1$, with $i \in \mathbb{N}$. Hence the maximum number of recursions needed until v strictly increases is the smallest x such that $k \leq 2^x - 1$, which is precisely R . ■

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

► **Theorem 3.6.** R-BST can be implemented in time $O(kn^2)$. ◀

Proof. Computing the level vectors of the optimal BSTs for each scenario requires time $O(kn^2)$ [Knu71]. From this, we can compute a vector M with $M[j] = \min_s L_j^s$, in time $O(kn)$. In addition, we use another table P , containing lists entries to keep track of the positions of each value in M . Formally, $P[v]$ is the list of all positions j with $M[j] = v$. Table P is constructed by a simple left-to-right processing of M , and as a result all the lists are sorted. During this process, we also create a table Q with ranks, such that $Q[j]$ is the number of entries in M between index 1 and j which has the value $M[j]$. P and Q are complementary in the sense that the i -th entry j in the list $P[v]$, is such that $M[j] = v$ and $Q[j] = i$. Finally, we store the arrays M and Q in a single *segment tree*, which allows to return for a given index interval $[i, j]$, the minimum value v in M between indices i and j , together with the ranks of the first occurrence of v and the last one in this interval. Let s, e be these ranks.

Line 6 of Algorithm 3 can be implemented by querying the segment tree at the interval $[i, j]$, and receiving v, s, e as a result. Then the algorithm chooses m as the $\lfloor (s + e)/2 \rfloor$ -th element in the list $P[v]$. Each query to the segment tree takes time $O(\log n)$. Since the resulting tree has n nodes, the construction of the tree takes

time $O(n \log n)$, which is also the pre-processing time of the segment tree. The total running time of the algorithm is therefore dominated by the time to build the level vectors L^s , i.e., at most $O(kn^2)$. ■

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

► **Theorem 3.7.** There exists a collection of k scenarios, described by vectors $F^1, \dots, F^k$, such that for every BST with level vector L , there exists a scenario s for which $\text{cost}(L, F^s) \geq \lceil \log_2(k+1) \rceil \text{cost}(L^s, F^s)$. ◀

Proof. Consider the following k scenarios $F^1, \dots, F^k$ on $n = k$ keys, for some k of the form $2^\ell - 1$, defined as

$$\begin{aligned} F^1 &= (1, 0, \dots, 0), \\ F^2 &= (0, 1, \dots, 0), \\ &\dots \\ F^k &= (0, 0, \dots, 1). \end{aligned}$$

Note that for each scenario, the corresponding optimal tree has cost 1. Since the complete tree with ℓ levels has $2^\ell - 1$ nodes, any tree with k nodes has at least one key at level $\lceil \log_2(k+1) \rceil$, and therefore cost at least ℓ in one of the scenarios. Hence, for every tree level vector L , there exists s such that $\text{cost}(L, F^s) \geq \lceil \log_2(k+1) \rceil \text{cost}(L^s, F^s)$. ■

3.3 Robust Huffman trees

3.3.1 Background and Measures

As discussed in Section 2.3, in the standard version of the Huffman tree problem, we are given n keys (for example, letters of an alphabet), along with a frequency vector that describes how frequently each key appears. The goal is to build a binary tree in which each key is assigned to a leaf, in such a way that the total cost $\sum_i F_i L_i$ is minimized, where L_i is the level of key i in the tree.

By labeling the edges leaving each internal node with 0 and 1, arbitrarily, we obtain a binary code: each key is assigned a codeword defined by the sequence of labels along the path from the root to its corresponding leaf. Since levels are assumed to start from 0, the level of a key equals the length of its codeword. The

resulting code is prefix-free, meaning no codeword is a prefix of any other. The optimal tree can be computed in time $O(n \log n)$ [Huf52].

We study Huffman coding in a robust setting in which we are given k n -dimensional frequency vectors $F^1, \dots, F^k$, 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 $F^s \cdot L = \sum_i F_i^s L_i$.

We can analyze robust HTs using the same measures as for robust BSTs, i.e., using cost-minimization, competitive ratio, or regret-based analysis; recall the discussion in Section 3.2.1. For Huffman trees, in particular, we will rely on regret-based analysis, which establishes more refined performance guarantees than competitive ratio; c.f. Corollary 3.12 which shows that our results for regret essentially carry over to the competitive ratio.

3.3.2 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 3.4). 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 [GJ79].

We first formulate the decision variant of the robust HT problem: Given two frequency vectors, F^1 and F^2 , and a threshold V , we must decide whether there exists a HT such that its cost does not exceed V in either scenario.

The following theorem relies on a proof that differs substantially from the corresponding NP-hardness proof for robust BSTs (Theorem 3.4). As discussed previously, these two problems differ significantly, primarily because the keys are located on the leaves in a Huffman tree. This structural difference forces us to establish entirely new properties to achieve the reduction and complete the NP-hardness proof. For the former, we need to use a different variant of the partition problem with equal cardinalities to satisfy all the conditions of the Huffman tree and achieve the desired construction.

► **Theorem 3.8.** 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. ◀

Proof. We first formulate the decision variant of the robust HT problem: Given two frequency vectors, F^1 and F^2 , and a threshold V , we must decide whether there exists a HT such that its cost does not exceed V in either scenario.

We will prove the result for the cost-minimization metric. At the end of the proof, we argue that it applies straightforwardly to the competitive ratio and the regret minimization as well.

The proof is based on a reduction from the EQUAL-CARDINALITY PARTITION (ECP) problem [GJ79]. An instance to this problem consists of a list $a_1, \dots, a_{2n}$ of non-negative integers with $\sum_i a_i = 2D$, and the goal is to decide whether there exists a binary vector $b \in \{0, 1\}^{2n}$ such that $\sum_i b_i a_i = \sum_i (1 - b_i) a_i = D$ and in addition $\sum_i b_i = n$. Without loss of generality, we can assume that n is of the form $n = 2^m$ for some integer m , as we can always pad the instance with zeros.

From the given instance of ECP, we define an instance of the robust HT problem, which consists of $3n + 1$ keys, a threshold V (to be determined later in the proof) and two scenarios. The keys are $v_1, \dots, v_{2n}$ and $u_1, \dots, u_n, u'$, having the following frequencies in the two scenarios:

frequency	v_i	u_j	u'
scenario 1	$M + a_i$	M	C_1
scenario 2	$M - a_i$	M	C_2

Here, C_1, C_2 and M are constants that will be specified later. Their role is to ensure that key u' is at level 1 for the robust HT, while also guaranteeing that the optimal trees for scenario 1 and for scenario 2 have the same cost. This can be achieved by a sequence of calculations. First, we can assume that both optimal trees have the same structure, meaning that key u' is at level 1 and there are exactly n keys at level $m + 2$ and $2n$ keys at level $m + 3$. We can achieve this by choosing a large enough constant M , which will also be specified later. Next, we focus on the constants C_1 and C_2 , assuming that the list of integers $a_1, a_2, \dots, a_{2n}$ is ordered from largest to smallest. We define the sums of the n smallest and n largest values as X and Y , respectively, and use them in the following expressions:

$$\text{cost}(L, F^1) = C_1 + M(2n(m + 3) + n(m + 2)) + (m + 2)Y + (m + 3)X, \text{ and}$$

$$\text{cost}(L, F^2) = C_2 + M(2n(m + 3) + n(m + 2)) - (m + 2)X - (m + 3)Y.$$

Since we know that $X + Y = 2D$, we obtain the desired property $\text{cost}(L, F^1) = \text{cost}(L, F^2)$ by setting $C_2 = C_1 + 2D(2m + 5)$. To simplify our notation, let us define $z := 2n(m + 3) + n(m + 2)$, as this expression will be used frequently later in the proof.

To establish the reduction, we will show that there is a HT on these $3n + 1$ keys with cost at most

$$V := C_1 + zM + D(2m + 5)$$

in each scenario if and only if there is a solution b to the partition problem.

For the one direction of the proof, let b be the solution to the partition problem. We construct a HT, in which for every $i = 1, \dots, 2n$, the key v_i is located at level $m + 2$ if $b_i = 0$ and at level $m + 3$ if $b_i = 1$. All n keys u_j are at level $m + 3$ and key u' is at level 1. See Figure 3.2 for an illustration.

In scenario 1 the cost of the tree is

$$\begin{aligned} C_1 + zM + (m + 2) \sum_i a_i(1 - b_i) + (m + 3) \sum_i a_i b_i &= \\ C_1 + zM + (m + 2) \sum_i a_i + \sum_i a_i b_i &= \\ C_1 + zM + (m + 2)2D + D. \end{aligned}$$

In scenario 2 the cost of the tree is

$$\begin{aligned} C_2 + zM - (m + 2) \sum_i a_i(1 - b_i) - (m + 3) \sum_i a_i b_i &= \\ C_1 + zM - (m + 2)2D - D. \end{aligned}$$

By the choice of C_2 , both costs are equal, and are also equal to V .

For the other direction of the proof, let L be the level vector of a HT with cost at most V in each scenario. First, we claim that key u' has to be at level 1 by the assumption that C_1 and C_2 are huge constants. Let u'' be the *sibling* of u' , i.e. the other node at level 1. Second, we claim that the sub-tree rooted at u'' is *complete*, in the sense that there are exactly $2n$ keys at level $m + 3$ and n keys at level $m + 2$. This structure, again as depicted in Figure 3.2, is imposed by the contribution of M in the frequencies as we argue next. Consider a Huffman tree T on $3n + 1$ keys, all with uniform frequencies M except for key u' which has frequency C . If T has the above mentioned structure, then it has cost $C + zM$. If u' is not at level 1, its cost is at least $2C$. Moreover, if u' is at level 1 and the sub-tree rooted at the sibling of u' is not complete then its cost is at least M higher. Choosing M large enough such that $M > D(2m + 5)$, we have that a tree with cost at most V is complete as well.

In addition, we can claim that there have to be exactly n leaves at level $m + 2$ and $2n$ leaves at level $m + 3$, since we know the structure of the tree from the optimality of the complete tree. By an exchange argument, we ensure that all u_j are at level $m + 3$, as their frequencies are strictly less than v_i in both scenarios. Let $b \in \{0, 1\}^{2n}$

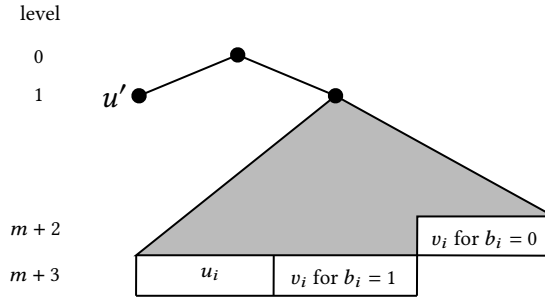


Figure 3.2: Schematic view of the NP-hardness proof construction for Theorem 3.8.

be a vector of indicator bits, with $b_i = 1$ if v_i is at level $m + 3$. The total cost of such a tree in scenario **1** is

$$C_1 + zM + (m + 2)2D + \sum a_i b_i,$$

while for scenario **2** its cost is

$$C_2 + zM - (m + 2)2D - \sum a_i b_i.$$

If both costs are at most V , then by the choice of C_2 they actually equal V . This implies that $D = \sum a_i b_i$, and b is a solution to the EQUAL-CARDINALITY PARTITION problem instance.

The above proof is stated in terms of the cost-minimization measure, however it applies straightforwardly to both the competitive ratio and regret minimization, since the optimal cost in both defined scenarios is the same, namely equal to V . ■

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 $i = 1, \dots, n$, it identifies the scenario s with the shortest code, denoted as $c_i^s \in \{0, 1\}^*$ (line 5). Next, it prepends exactly $\lceil \log_2 k \rceil$ bits to each code c_i^s , which represent the scenario s (lines 6 and 7). The algorithm generates the final HT by associating each key with a level equal to $\lceil \log_2 k \rceil + c_i^s$ (line 8). In line 9, “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 3.9.** Algorithm R-HT outputs a tree T with a valid level vector L of regret at most $\lceil \log_2 k \rceil$. That is, $F^s \cdot L \leq \min_{L^*} F^s \cdot L^* + \lceil \log_2 k \rceil$, for every scenario s . ◀

Algorithm 4: R-HT

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

```

1 foreach scenario  $s \in \{1, \dots, k\}$  do
2   └ Let  $T^s$  be the Huffman tree of minimum cost for frequency vector  $F^s$ ;
3 Let  $C \leftarrow \emptyset$ ;
4 foreach key  $i \in \{1, \dots, n\}$  do
5   └ Let  $s$  be a scenario for which key  $i$  has the shortest codeword  $c_i^s \in \{0, 1\}^*$ ;
6   └ Let  $x$  be the binary encoding of the integer  $s - 1$  using exactly  $\lceil \log_2 k \rceil$ 
      bits;
7   └ Add the codeword  $x \cdot c_i^s$  to  $C$ ;
8 Build the Huffman tree  $T$  for the prefix-free codewords in  $C$ ;
9 Compactify  $T$ ;
10 return  $T$ 

```

Proof. First, we observe that the set of all s binary encodings of the integers $0, \dots, k-1$ form a prefix-free code. Moreover, we observe that concatenating these strings with subsets of the prefix codes for each scenario yields a code which is again prefix-free.

By construction, for every key i , the length of its resulting code does not exceed by more than $\lceil \log_2 k \rceil$ the length of the optimal code among all scenarios. This clearly also holds after compactification, hence the theorem follows. ■

► **Proposition 3.10.** Algorithm R-HT can be implemented in time $O(kn \log n)$, i.e., its running time is dominated by the time required to find optimal HTs for each scenario. ◀

Proof. For an efficient implementation of R-HT, one can work with vectors of code lengths. If L_i^s is the length of the codeword for key i in the optimal HT for scenario s , then before compactification, we construct a code length vector L with $L_i = \lceil \log_2 k \rceil + \min_s L_i^s$ for every key i . Then, we build the actual HT level by level, assigning for each level ℓ the keys i with $L_i = \ell$ to nodes of the growing tree. The construction is done in time linear in the size of the resulting tree. ■

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

► **Theorem 3.11.** There exists a set of k -scenarios for which no robust HT has regret smaller than $\lceil \log_2 k \rceil$. ◀

Proof. Consider k scenarios defined by frequency vectors $F^1, \dots, F^k$ on $n = k + 1$ keys, with $F_i^s = 1$ if $s = i \in \{1, \dots, n\}$ and $F_i^s = 0$ otherwise. Note that $k + 1$ is a dummy key which has zero probabilities in every scenario, but will be useful in the construction. Clearly, we have that $\min_{L^*} F^s \cdot L^* = 1$ for every scenario s . By a simple counting argument, every valid level vector on n keys must assign level at least $\lceil \log_2 n \rceil$ to at least one key. Moreover, since inner nodes have out-degree 2, there must exist at least two keys at level at least $\lceil \log_2 n \rceil$. Let s be one of those keys, with the property $s \in \{1, \dots, k\}$, and with a slight abuse of notation, let s also denote the corresponding scenario (recall that each scenario has support 1 in the given construction). It follows that the cost of the tree for scenario s is at least $\lceil \log_2 n \rceil$, and thus the difference from the optimal cost of 1 for scenario s is at least $\lceil \log_2 n \rceil - 1$. If k is a power of 2 we have

$$\lceil \log_2 n \rceil - 1 = \log_2 k + 1 - 1 = \lfloor \log_2 k \rfloor,$$

whereas if k is not a power of two we also have

$$\lceil \log_2 n \rceil - 1 = \lceil \log_2 k \rceil - 1 = \lfloor \log_2 k \rfloor,$$

which completes the proof. ■

We conclude this section by noting that the guarantees we established on the regret of HTs essentially carry over to the competitive analysis of the trees.

► **Corollary 3.12.** 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 $\lfloor \log k \rfloor + 1$. ◀

Proof. To establish a competitive ratio of $\lceil \log_2 k \rceil + 1$ for R-HT, we can directly apply the proof of Theorem 3.9, along with the fact that the cost of the optimal HT for any scenario s is at least 1. The lower bound follows by the same adversarial construction as in the proof of Theorem 3.11, replacing the differences by the appropriate ratios at the conclusion of the proof. ■

3.4 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. 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, as shown in the following proposition.

► **Proposition 3.13.** A worst case instance for the robust BST problem satisfies that every key i is dedicated to a specific scenario, in the sense that it has a positive probability for at most one scenario. ◀

Proof. Let I be an instance with 2 scenarios. Let i be a key in I . Let p^0, p^1 be the distributions for the respective scenarios **0** and **1**.

Let I' be an instance obtained from I by replacing key i by new keys i', i'' and with the following probabilities

	i'	i''
scenario 0	p_i^0	0
scenario 1	0	p_i^1

Observe that in the optimal tree for scenario **0** in I' , the key i'' must be a leaf by optimality. Hence, it has the same cost as the optimal tree for scenario **0** in I . The same observation holds for scenario **1**.

Let T' be a BST which minimizes the competitive ratio for I' . Suppose that the level of i' is at most the level of i'' in T' . The proof for the other case is symmetric.

Observe that i'' has no left descendant in T' . The reason is that this left descendant needs to be i' or its ancestor by the key ordering. But i' is at the same or lower level than i'' .

Now transform T' into a tree T for instance I as follows. Replace i' by i . Remove i'' if it is a leaf, otherwise replace i'' by its unique descendant. The level of each

key different from i, i', i'' is decreased by 1 for the descendants of i'' in T' and preserved otherwise. In scenario **0** of instance I' , the keys i' and i'' contribute with the value which is at most the value with which key i contributes in scenario **0** of instance I . This is due by the level ordering assumption between i' and i'' . The same observation holds for scenario **1**.

As a result in each scenario, the cost of T' is at least the cost of T , which is at least the optimal cost. It follows that the competitive ratio for I' is at least the competitive ratio for I . Hence the worst case instance for the competitive ratio satisfies without loss of generality that each key has a positive probability for exactly one scenario. The same observation holds for the cost minimization and the regret minimization variants. ■

There are two important observations to make. First, unlike the approaches of Section 3.2 and Section 3.3, the fairness setting 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* [BV04] 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 $1, \dots, n$. 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.

► **Remark 3.14.** Note that we do not normalize frequencies in this setting; as a result, the cost and regret values are integers. This design choice is important, as it allows us to use integer-valued parameters in the dynamic programming formulation introduced in Section 3.5.1. ◀

To define the concept of regret, let $\text{OPT}(m)$ denote the optimal cost of a binary search tree over m keys under a uniform key distribution. From [Knu97, Sect. 5.3.1, Eq. (3)], the following closed-form expression holds:

$$\text{OPT}(m) = (m + 1) \lceil \log_2(m + 1) \rceil - 2^{\lceil \log_2(m + 1) \rceil} + 1. \quad (3.2)$$

This expression is derived from the structure of an optimal binary search tree for uniformly distributed keys, which is a *complete binary tree*. In such a tree, all levels except possibly the last are completely filled, and all nodes on the last level are aligned as far to the left as possible. The cost of accessing a key is defined as its depth in the tree, and the total cost is the sum of all access costs across the m keys.

Due to the uniform distribution, the structure of the tree is balanced to minimize average depth. The cumulative cost can thus be interpreted as a sequence counting node levels: starting from 1 at the root, the number of nodes at each level doubles until the last level, producing a depth pattern of the form

$$1, 2, 2, 3, 3, 3, 3, 4, 4, \dots,$$

where each depth level ℓ contributes $2^{\ell-1}$ times to the cost. Surprisingly, this same quantity also corresponds to the worst-case number of comparisons needed to insert m elements into a binary search tree [Knu71], confirming the tight connection between optimal tree structure and binary search cost.

Clearly, in every BST T , the 0-cost is at least $\text{OPT}(a)$. We refer to the difference between the 0-cost of T and the quantity $\text{OPT}(a)$ 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 3.15.** 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. This is illustrated in Figure 3.3.

3.4.1 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 $\alpha = 0, 1, \dots$, until $f(s, \alpha) = 0$. We first need to bound the range of α .

► **Lemma 3.16.** Let α^* denote the smallest integer such that $f(s, \alpha^*) = 0$. It holds that $\alpha^* \leq a \lfloor \log_2(b+2) \rfloor$. ◀

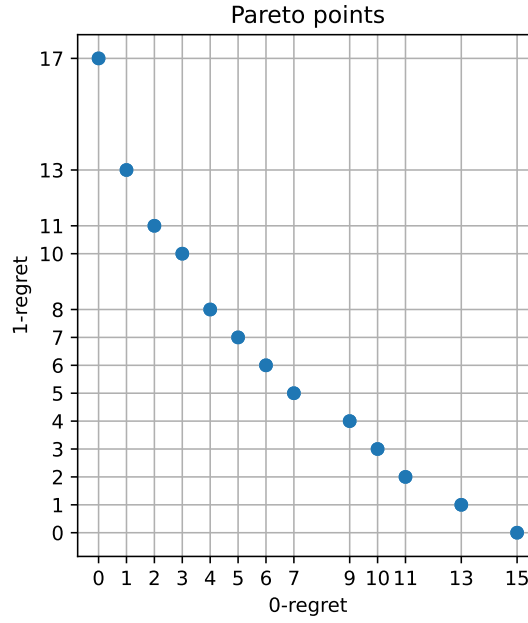


Figure 3.3: The Pareto optimal regret points for the string 1011011001111000.

Proof. Any BST with zero 1-regret consists of a complete tree for the scenario 1 keys. Such a tree has depth $d = \lfloor \log_2(b+1) \rfloor$, where b is the number of 1s in s . There are three types of nodes in this tree, depending on their out-degree: leaves, nodes with out-degree 1, and full nodes. All nodes strictly below level $d-1$ are full. All nodes at level d are leaves. At level $d-1$, there may be nodes with out-degree 1, moreover, there may be leaves at level $d-1$ if and only if $b \leq 2^d - 3$. Hence, a worst-case string s is $10^a 1^{b-1}$, for which the 0-keys need to form a complete tree, and which is attached to the right of the left-most 1-key. This 1-key can be at level $d-1$ only if $b \leq 2^d - 3$. As a result, $\alpha^* = a \lfloor \log_2(b+2) \rfloor$. Figure 3.4 illustrates this situation. ■

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

$$\text{loss}(m_1, m_0, m_2) = m_1 + m_0 + m_2 + \text{OPT}(m_1) + \text{OPT}(m_2) - \text{OPT}(m_1 + m_0 + m_2),$$

and recall that OPT is given by (3.2).

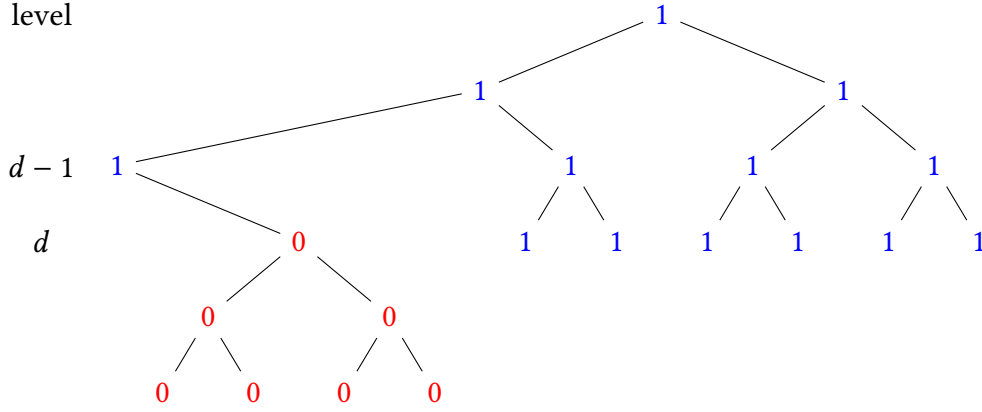


Figure 3.4: An illustration of the situation in the proof of Lemma 3.16, for deriving α^* .

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 $\text{loss}(m_1, m_0, m_2)$.

We show how to compute the function f using dynamic programming. While one could use the approach of [GMS04], 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

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

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 $\alpha = \alpha_1 + \text{loss}(a_1, a_0, a_2) + \alpha_2$.

► **Theorem 3.17.** The function $f(s, \alpha)$ can be computed using the above dynamic programming algorithm. The overall running time is $O(n^5 \log^2 n)$. ◀

Proof. The correctness of the algorithm follows from the recursive structure of

binary search trees. Any BST for s with 0-regret at most α is defined by a choice of root r , and a partition of the residual regret budget $\alpha - \text{loss}(a_1, a_0, a_2)$ between the left and right subtrees. The subtrees correspond to sub-strings $s[1 : r - 1]$ and $s[r + 1 : n]$, and are handled recursively under the corresponding regret bounds α_1 and α_2 , satisfying $\alpha_1 + \alpha_2 = \alpha - \text{loss}(a_1, a_0, a_2)$. This directly yields the recurrence defining f .

For the running time, observe that there are $O(n^2)$ distinct sub-strings of s , and by Lemma 3.16, at most $O(n^2 \log n)$ values of α to consider for each sub-string. For each (s, α) pair, the algorithm considers $O(n)$ choices for the root position r , and for each such r , it considers $O(\alpha^*) = O(n \log n)$ possible splits of the regret bound. The total number of subproblems is $O(n^2 \alpha^*)$, and each subproblem is solved in $O(n \alpha^*)$ time. Therefore, the overall time complexity is $O(n^3 (\alpha^*)^2) = O(n^5 \log^2 n)$. ■

► **Theorem 3.18.** The dynamic programming algorithm for computing the function f can be generalized to $k \geq 2$ scenarios. The resulting algorithm runs in time $O(n^{1+2k} \log^{2k-2} n)$. ◀

Proof. In the case of k scenarios, the regret of a tree is represented as a k -dimensional vector. The function f is evaluated over k -tuples $(s, \alpha_1, \dots, \alpha_{k-1})$, where $\alpha_1, \dots, \alpha_{k-1}$ are fixed regret bounds for the first $k - 1$ scenarios, and the function minimizes the remaining regret dimension.

By Lemma 3.16, each α_i ranges over $O(n \log n)$ values. As there are $O(n^2)$ distinct sub-strings s , the total number of subproblems is $O(n^2 (n \log n)^{k-1}) = O(n^{1+k} \log^{k-1} n)$.

To compute each entry of f , the algorithm considers $O(n)$ choices for the root position and $O((n \log n)^{k-1})$ ways to partition the regret bounds for the $k - 1$ dimensions among the two subtrees. Therefore, each subproblem is solved in $O(n(n \log n)^{k-1})$ time. The total running time is thus

$$O(n^{1+k} \log^{k-1} n) \cdot O(n(n \log n)^{k-1}) = O(n^{1+2k} \log^{2k-2} n).$$

■

► **Remark 3.19.** The dynamic program presented in this section assumes uniform probabilities for the keys dedicated to each scenario. This is essential because the function f takes as input a bound on the 0-regret, which is defined as the difference in costs between scenarios. ◀

3.5 Computational Experiments

3.5.1 Robust BSTs and HTs

We report computational experiments on the robust versions of BSTs and HTs from Section 3.2 and Section 3.3. We used open data from [Wik24]. Specifically, we chose ten European languages¹ 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%.

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

Table 3.1: Performance comparison of the various algorithms.

To evaluate our algorithms, we provide a Mixed Integer Linear Program (MILP) formulation for the robust BST problem. This allows us to compute optimal trees using commercial MILP solvers such as GUROBI. We present the formulation for the cost-minimization objective; the competitive ratio and regret variants follow analogously by modifying only the objective function. The index ranges are $i, j, \ell \in \{1, \dots, n\}$.

$$\text{minimize } C \tag{3.3}$$

$$\text{subject to } \forall i : \sum_{\ell} x_{\ell,i} = 1 \tag{3.4}$$

$$\forall i, j, \ell : \sum_{r=i+1}^{j-1} \sum_{u=1}^{\ell-1} x_{u,r} \geq x_{\ell,i} + x_{\ell,j} - 1 \tag{3.5}$$

$$\forall s : \sum_{\ell} \sum_i F_i^s \cdot \ell \cdot x_{\ell,i} \leq C \tag{3.6}$$

¹ Danish, Dutch, English, Finnish, French, German, Italian, Portuguese, Spanish and Swedish.

$$\forall i, \ell : x_{\ell,i} \in \{0, 1\} \quad (3.7)$$

Each variable $x_{\ell,i}$ is binary and equals 1 if and only if key i is assigned to level ℓ in the tree.

- Constraint (3.4) ensures that each key is placed at exactly one level. Formally, it encodes a partition of the keys across levels, with each key having exactly one active level assignment.
- Constraint (3.5) enforces the structural validity of the tree. Specifically, it implements the requirement of Definition 3.1: if two keys i and j are assigned the same level ℓ , then at least one key r strictly between them (i.e., $i < r < j$) must be assigned to a level $u < \ell$.
- Constraint (3.6) guarantees that the cost of the tree under each scenario s is at most C , where the cost is computed as the expected depth of access across keys, weighted by their scenario-specific frequencies F_i^s .
- The objective (3.3) then minimizes C , i.e., the maximum cost over all scenarios.

To model the competitive ratio and regret variants, we retain the same objective function (3.3), and modify only the right-hand side of constraint (3.6).

- For the competitive ratio variant, we assume that for each scenario s , the optimal level assignment $(L_i^s)_i$ is precomputed. Then constraint (3.6) is replaced by

$$\forall s : \sum_{\ell} \sum_i F_i^s \cdot \ell \cdot x_{\ell,i} \leq C \cdot \sum_i F_i^s \cdot L_i^s.$$

- For the regret variant, using the same precomputed levels L_i^s , constraint (3.6) is replaced by

$$\forall s : \sum_{\ell} \sum_i F_i^s \cdot \ell \cdot x_{\ell,i} - \sum_i F_i^s \cdot L_i^s \leq C.$$

In both cases, the quantity C is minimized and represents either the worst-case competitive ratio or the worst-case regret over all scenarios.

The MILP for Huffman Trees (HTs) can be derived through a similar process, with a few modifications. First, levels ℓ start at 0 rather than 1. Second, constraint (3.5) is replaced by a linear condition that guarantees the resulting level assignment corresponds to a valid prefix-free code.

Let L_i denote the level assigned to key i . Then for any level ℓ , the number of keys at level $b \leq \ell$ must satisfy:

$$\sum_{b=0}^{\ell} 2^{\ell-b} \cdot |\{i : L_i = b\}| \leq 2^{\ell}.$$

This condition ensures that there exists a prefix-free binary code with the given level assignments.

We rewrite this as a linear constraint using the binary assignment variables $x_{b,i}$:

$$\sum_{b=0}^{\ell} 2^{\ell-b} \sum_i x_{b,i} \leq 2^{\ell} \quad \forall \ell.$$

This constraint replaces (3.5) in the MILP for HTs. All other constraints (assignment, cost, binary domain) remain unchanged.

We let Gurobi Optimizer version 9.1.2 solve our MILPs on a standard Windows 11 laptop with an Intel(R) Core(TM) i5-8365U CPU processor running at 1.60GHz with 16Gb of on-chip RAM. The observed running times, in seconds, are shown in the following Table 3.2.

	Binary Search Tree	Huffman Tree
cost	8.23	0.07
competitive ratio	14.47	0.12
regret	13.78	0.04

Table 3.2: GUROBI running times in seconds.

In contrast, R-BST and R-HT run in less than 0.1 seconds, using a Python implementation. Table 3.1 summarizes the results of the experiments. As expected, the empirical performance is better than the worst-case guaranties of Theorem 3.5 and Theorem 3.9. This is due to the fact that real data do not typically reflect adversarial scenarios (compare, e.g., to the adversarial constructions of Theorem 3.7 and Theorem 3.11).

3.5.2 Pareto-Optimality and Fairness

We report experiments on our Pareto-optimal algorithm of Section 3.4, denoted by PO. The universe of all possible keys is represented by the names of cities from 10 different countries (the same countries as in Section 3.5.1). From this data, a string s

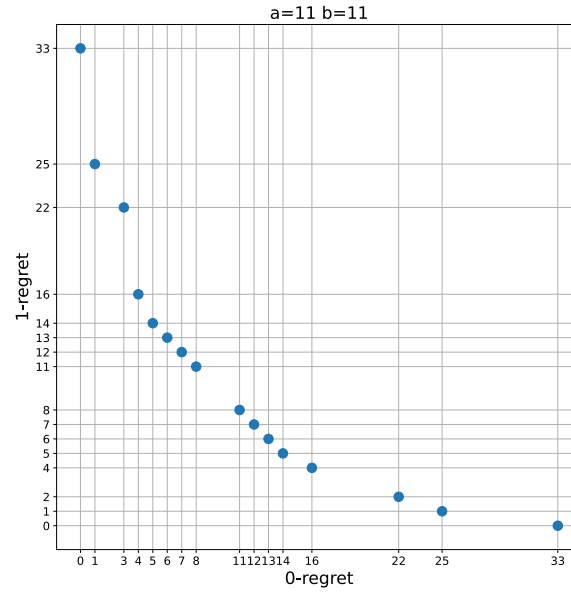


Figure 3.5: The Pareto front for strings with $a = 11, b = 11$.

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 $n = 30$. 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 $a, b \in \{0, \dots, 11\}$, 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 3.5 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 $b \leq 11$, as summarized in the following observation.

► **Conjecture 3.20.** 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 . ◀

We observe that one can get quite close to these regret bounds in practice, as will be detailed in the next section.

Despite the strong empirical evidence supporting this conjecture, we were not able to establish an analytical proof. The main difficulty lies in the recursive nature of BSTs, and the fact that the optimal solution does not have an obvious structure that one may exploit. To illustrate this difficulty, we discuss below an instance for which the conjecture holds, but not “locally”, i.e., it does not hold for all subtrees.

Consider strings of the form $01^k 0^s 1$ for $k, s \geq 6$. In all BSTs that satisfy the conjecture, we observe that in some subtrees the local regret for one of the types exceeds the corresponding local budget. In other words, although the overall regret may meet the conjectured bound, the regret for 0s or 1s within a subtree can exceed the number of nodes of that type.

Figure 3.6 illustrates this behavior. The root of the tree is a 1. The left subtree forms a complete tree of 0s (with the exception of a 1 as the leftmost leaf), and the right subtree is a complete tree of 1s (with a single 0 as the rightmost leaf). In both subtrees, the local regret for one of the classes equals 2, which exceeds the conjectured bound.

This observation suggests that the conjecture cannot be proved using a recursive construction or a standard inductive argument, despite the recursive nature of binary search trees. Instead, a complete proof appears to require a full understanding of the entire Pareto curve.

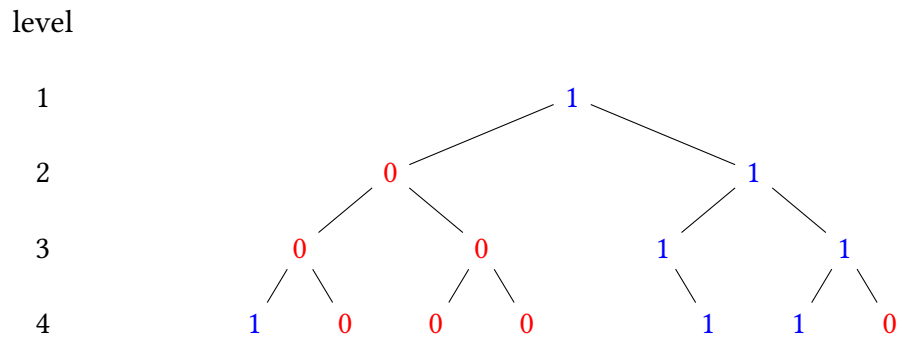


Figure 3.6: An example for strings of the form $01^k 0^s 1$ with $k, s \geq 6$. The depicted tree satisfies the global regret bounds, but the subtrees rooted at level 2 locally exceed the regret bound for one of the two types.

3.6 Relation to the Entropy

In this section, we discuss the connection between the notion of entropy and the structure of binary search trees. This will help us to prove a result that is very close to Conjecture 3.20. This section builds on the background provided in Section 2.3.1.

Let p be a probability distribution over the n keys, and let $H(p)$ denote its entropy, defined as

$$H(p) = \sum_{i=1}^n p_i \log \frac{1}{p_i},$$

where all logarithms are in base 2. In an optimal binary search tree for p , the expected cost is minimized, and a natural question is how this cost relates to $H(p)$.

Mehlhorn [Meh75] proposed a simple recursive rule for building near-optimal BSTs:

Rule II (Bisection): Choose the root so that the total weight (frequency) of the left and right subtrees is as balanced as possible, and apply the rule recursively.

This construction results in a binary search tree, denoted by BB, that can be built in $O(n \log n)$ time. It satisfies the property that the level of each key i is at most $\lceil \log(1/p_i) \rceil$, with the root at level 0. Consequently, the total expected cost of the BB tree satisfies

$$\text{BB} \leq H(p) + 1.$$

► **Remark 3.21.** Throughout this section, we adopt the convention that the root of the BST is at level 0, as is standard in analyses involving entropy and Mehlhorn's BB construction. This is in contrast to previous sections, where the root was defined to be at level 1. That earlier choice was so as to avoid pathological instances where only one key contributes to the cost, and would thus be the root of the tree. ◀

Lower bounds on the optimal cost OPT are also known [PS96], namely:

1. $\text{OPT} \geq \frac{H(p)}{\log 3}$, based on the connection between BSTs and ternary Huffman codes.
2. $\text{OPT} \geq H(p) + 1 - \log e - \log \log(n + 1)$.
3. $\text{OPT} \geq H(p) + H(p) \log H(p) - (H(p) + 1) \log(H(p) + 1)$.

Additionally, Mehlhorn [Meh84, Chapter III, Theorem 9, p. 182] provides the following approximation guarantee:

$$\text{BB} - \text{OPT} \leq \log \text{OPT} \approx \log H(p) \leq \log \log n.$$

This bound suggests that if OPT were equal to $H(p)$, then Conjecture 3.20 would follow. Although this is not the case, we can still interpret the BB construction as guaranteeing a regret of at most $O(\log \log n)$, as we show in Theorem 3.22.

► **Theorem 3.22.** Let F^1 and F^2 be two frequency vectors over n keys, and let $\bar{F} = (F^1 + F^2)/2$. Let L^1 and L^2 be the optimal level vectors minimizing $\text{cost}(L, F^1)$ and $\text{cost}(L, F^2)$ respectively. Let L_{bb} be the level vector produced by the BB construction applied to $\bar{F}$. Then:

$$\text{cost}(L_{\text{bb}}, F^1) \leq \text{cost}(L^1, F^1) + O(\log \log n),$$

and similarly for F^2 . ◀

Proof. By construction, the tree output by BB on a given frequency vector ensures that the level of each key i is at most $\lfloor \log(1/\bar{F}_i) \rfloor$. This implies that:

$$L_{\text{bb},i} \leq \log\left(\frac{2}{F_i^1 + F_i^2}\right).$$

We can then bound the cost under F^1 by:

$$\text{cost}(L_{\text{bb}}, F^1) = \sum_{i=1}^n L_{\text{bb},i} \cdot F_i^1 \leq \sum_{i=1}^n \log\left(\frac{2}{F_i^1 + F_i^2}\right) \cdot F_i^1.$$

For any i , we have:

$$\log\left(\frac{2}{F_i^1 + F_i^2}\right) \leq \log\left(\frac{1}{F_i^1}\right) + 1.$$

Using this, we obtain:

$$\text{cost}(L_{\text{bb}}, F^1) \leq \sum_{i=1}^n \left(\log\left(\frac{1}{F_i^1}\right) + 1 \right) \cdot F_i^1 = H(F^1) + \sum_{i=1}^n F_i^1.$$

Assuming F^1 is normalized to sum to 1, the final expression becomes:

$$\text{cost}(L_{\text{bb}}, F^1) \leq H(F^1) + 1.$$

From known bounds (see [Meh84, Chapter III, Theorem 9, p. 182]), we have:

$$\text{cost}(L^1, F^1) \geq H(F^1) - O(\log \log n),$$

which gives:

$$\text{cost}(L_{\text{bb}}, F^1) \leq \text{cost}(L^1, F^1) + O(\log \log n).$$

This completes the proof. ■

This result shows that although the BB tree may not always minimize the regret, it incurs an additive term $O(\log \log n)$ compared to the best possible tree for either F^1 or F^2 . This is the closest upper bound we can prove towards Conjecture 3.20.

The same idea can be extended to more than two scenarios. When taking the average of k frequency vectors and applying the BB construction, the level of each key increases by at most an additive term $\log k$. Therefore, the regret is bounded by $O(\log k) + O(\log \log n)$ across all k scenarios.

3.7 Conclusion

In this chapter, we introduced the study of scenario-based robust optimization applied to fundamental data structures, specifically Binary Search Trees and Huffman Trees. We analyzed models in which the input consists of multiple scenarios, each given by a frequency distribution, and the goal is to construct a single tree that performs well across all scenarios.

Our contributions include hardness results, as well as optimal algorithms under various robustness measures, such as competitive ratio, regret, and Pareto-optimality. In particular, we proposed efficient dynamic programming techniques and formalized the connection between robust optimization and multi-objective regret minimization. We also explored how fairness can be captured through Pareto-optimality, providing a conceptual bridge between algorithmic fairness and robust design.

Several directions remain for future work. One natural extension is to investigate other hierarchical data structures such as B-trees or quad-trees within the same robustness framework. Another is to develop approximation algorithms for NP-hard variants, such as minimizing the worst-case cost over all scenarios. More broadly, the trade-off between structure cost and access frequency that we studied

here is central to many applications, including scheduling, caching, and resource management problems, such as inventory organization in warehouses.

The techniques developed in this chapter form part of the broader goals of this thesis, which combines classical algorithm design with ideas from robust optimization and fairness. They contribute to a general framework for building data structures that perform reliably across multiple scenarios and under uncertainty.

4.1 Introduction

Prophet inequalities is a well-studied problem in online decision-making. This problem involves a setting in which values from known distributions are revealed one by one, and the algorithm must decide immediately whether to accept the current value or reject and move on, without knowing what future values will be. The goal is to perform nearly as well as a “prophet” who knows all the values in advance and can always choose the best one. For the formal definitions and background, we refer the reader to Section 2.1.3.

As discussed in Section 2.1.3, the standard setting assumes full knowledge of the underlying distributions on the values. While this assumption is convenient in theory, it is often unrealistic in practice. In many real-world scenarios, the decision-maker may only have partial information or must deal with uncertainty about the actual distributions.

In this chapter, we study a more general model in which the algorithm is given a set of possible scenarios, each representing a different set of distributions. The true (actual) scenario is one of these possible scenarios. This setup captures, for instance, situations where the distributions depend on an unknown parameter, and each scenario corresponds to one possible realization of that parameter.

We are particularly interested in strategies that can adapt to this kind of uncertainty. Our goal is to develop algorithms that perform well across multiple scenarios, even if the exact one is not known. In addition, we explore how some form of learning or partial observation during the execution of the algorithm can help improve decisions in practice.

4.1.1 Further Related Work

We discuss some related results beyond those covered in Section 2.1.3.

A large body of work has extended the prophet inequality to more complex settings. This includes selecting multiple items [EFN18], and sets under matroid constraints [Düt+20; KW12], unknown distributions [AKW19], and handling cases where items arrive in continuous time [All06; KK18]. Fairness constraints have

also been studied [AK22; Cor+21]. General surveys on these topics can be found in [Cor+18; HK92; Luc17].

Recent work has also explored the use of *advice*, particularly via sampling. This was studied in [Dia+21], where the algorithm learns the distributions using sampling. Given a sufficient number of samples, the resulting algorithm is both robust and consistent. Roughly speaking, the algorithm performs better as the number of samples increases, and has worst-case guarantees if the samples are adversarially generated. Brustle et al. [Bru+24a; Bru+24b] analyze a model where the algorithm handles multiple independent draws from the distribution and compares its performance to the optimal offline value. This resource augmentation model has applications in pricing and mechanism design. More recently, Ezra et al. [EG24] extended this approach to the much more complex setting in which there is correlation among the box distributions.

4.2 Model

As we recall from Section 2.1.3, in the prophet inequality problem, an algorithm observes a sequence of n random variables $X_1, \dots, X_n$ drawn independently from n known distributions and must decide, one by one, whether to accept the current value or move on. In this chapter, we consider the model in which there are k possible scenarios, where each scenario $s \in \{1, \dots, k\}$ defines a sequence of distributions $D_1^s, \dots, D_n^s$. We write $X^s = (X_1^s, \dots, X_n^s)$ where each X_i^s is drawn independently from D_i^s . The true scenario is fixed, but unknown to the algorithm.

We evaluate the performance by the use of the competitive ratio:

$$CR = \min_{s \in [k]} \frac{\mathbb{E}[A(X^s)]}{\mathbb{E}[\max(X^s)]}, \quad (4.1)$$

where $A(X^s)$ is the reward of algorithm A under scenario s . The goal is to find an algorithm that maximizes this worst-case ratio across all scenarios.

► **Remark 4.1.** In this chapter, we use the convention that the competitive ratio is the worst-case ratio between the algorithm's expected reward and that of the prophet, hence at most 1. This differs from the convention we used in Section 2.1.3, where the competitive ratio of a profit maximization problem is the reciprocal ratio, hence always at least 1. We follow this convention since it is commonly used in the study of prophet inequalities, and makes it easier to compare to known results in the literature. ◀

There are several known strategies in the single-scenario setting. One of them is the algorithm of Hill and Kertz [HK81], which computes a different threshold for each step using dynamic programming and backward induction. This algorithm, to which we refer as the HK algorithm, works as follows. Define

$$T_i = \mathbb{E}[\max(X_{i+1}, T_{i+1})],$$

where $T_n = 0$, then T_i can be computed recursively from the above relation. This recursive rule ensures that T_i maximizes the expected reward over the values in boxes $1 \dots i$, assuming an optimal strategy from position $i + 1$ onward. The computed value of T_0 is then the total expected reward obtained by the algorithm.

In the remainder of this chapter, we will consider both fixed-threshold strategies and the HK algorithm as starting points, and investigate how they can be adapted to the multi-scenario setting. We also introduce new strategies that aim to handle uncertainty more effectively, especially in situations arising in practice.

4.3 A Tight Upper Bound on the Competitive Ratio

We begin by showing a tight negative result on the competitive ratio.

► **Theorem 4.2.** For any given integer $k \geq 1$, there exists an instance involving k scenarios such that for every strategy A , there is a scenario s for which

$$\frac{\mathbb{E}[A(X^s)]}{\mathbb{E}[\max(X^s)]} \leq \frac{1}{2k}.$$

◀

Proof. We will first show a simpler, but weaker, upper bound construction, establishing $\text{CR} \leq 1/k$.

The instance consists of k boxes with deterministic values, represented as a matrix with scenarios as rows and boxes as columns:

$$\begin{pmatrix} M & 0 & \dots & 0 \\ M & M^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ M & M^2 & \dots & M^k \end{pmatrix}$$

where $M \gg 1$ is a large constant. In scenario s , the maximum value is $\max(X^s) = M^s$.

Let A be any strategy, and let p_i be the probability that A selects the i -th value. Since only one value is selected,

$$\sum_{i=1}^k p_i \leq 1.$$

In scenario s , the expected reward of A is at most

$$\mathbb{E}[A(X^s)] \leq p_s M^s + M^{s-1}$$

Dividing by the optimal reward in scenario s yields:

$$\frac{\mathbb{E}[A(X^s)]}{\max(X^s)} \leq p_s + \frac{1}{M}.$$

Hence the competitive ratio of A in scenario s tends to p_s , as $M \rightarrow \infty$. Thus, it is optimal for strategy A to set all these probabilities to $1/k$, establishing the claimed bound.

To obtain a better bound of $\frac{1}{2k}$, we modify the above construction by duplicating each box. Specifically, each scenario s now consists of k pairs of boxes: the first box in the j -th pair has the same value as the j -th box in the previous construction. More precisely the first box of the j -pair is M^j with probability 1, and the second box in the j -th pair has value M^{j+1} with probability $\frac{1}{M}$, and 0 otherwise. We illustrate these distributions by the following matrix, where $M^i?$ means that the box value is M^i with probability $1/M$ and 0 otherwise:

$$\begin{pmatrix} M & M^2? & 0 & 0 & \dots & 0 & 0 \\ M & M^2? & M^2 & M^3? & \dots & 0 & 0 \\ \vdots & & & & \ddots & 0 & 0 \\ M & M^2? & M^2 & M^3? & \dots & M^k & M^{k+1}? \end{pmatrix}.$$

In this construction, even-indexed boxes are “randomized”, while odd-indexed boxes have a “deterministic” value.

First, we show that no strategy ever selects an even-indexed box. By contradiction, consider the event that A accepts box $2j$ conditioned on its value being M^{j+1} , and let q_j denote the probability of this event. This event contributes

$$q_j \cdot \mathbb{P}(X_{2j}^s = M^{j+1}) \cdot M^{j+1} = q_j \cdot \frac{1}{M} \cdot M^{j+1} = q_j M^j$$

to the expected reward in every scenario $s \geq j$. Hence, in terms of expected reward, it is equivalent to accept box $2j - 1$ instead, and move the probability mass q_j to stopping at box $2j - 1$, which has value M^j deterministically. This modification contributes

$$q_j \cdot M^j$$

to the expected reward in every scenario $s \geq j$, and therefore does not decrease the expected reward in any scenario.

As a result, we can assume that the strategy accepts only deterministic boxes, and the analysis of its expected reward is the same as in the previous, simpler construction. Let p_i be the probability of selecting box $2i - 1$ (i.e., the i -th deterministic box). In scenario s , the expected reward is at most

$$\mathbb{E}[A(X^s)] \leq p_s M^s + M^{s-1}.$$

However, in this modified instance, the expected reward of the prophet in scenario s is

$$\mathbb{E}[\max(X^s)] = 2M^s - M^{s-1}.$$

Consequently, for M tending to infinity, the upper bound on the competitive ratio tends to $1/(2k)$

■

The above result shows that the worst-case performance degrades linearly with the number of scenarios. Note that the construction is designed so that any algorithm cannot identify the scenario based on early observations. This is because, once a zero value is observed, it is already too late for the algorithm, and the opportunity to get a high value is gone.

4.4 A Baseline Strategy

The following is a simple strategy that matches the upper bound of $\frac{1}{2k}$ from Theorem 4.2: choose a scenario $s \in \{1, \dots, k\}$ uniformly at random and apply the optimal strategy for that scenario, as if it were the true one. In particular, we apply the HK algorithm to scenario s . We refer to this strategy as the algorithm UNIFORM.

► **Definition 4.3.** The algorithm UNIFORM commits to a scenario $s \in \{1, \dots, k\}$ chosen uniformly at random, and applies the HK strategy for that scenario without updating its choice based on observed values. ◀

► **Theorem 4.4.** Algorithm UNIFORM achieves a competitive ratio of at least $\frac{1}{2k}$. ◀

Proof. Fix an instance with k scenarios. For any scenario $s \in \{1, \dots, k\}$, let $X^s = (X_1^s, \dots, X_n^s)$ denote the sequence of random variables under scenario s . Let A_s denote the HK algorithm applied to scenario s , which thus satisfies:

$$\mathbb{E}[A_s(X^s)] \geq \frac{1}{2} \cdot \mathbb{E}[\max(X^s)].$$

Since UNIFORM chooses each scenario with probability $1/k$, its expected reward in scenario s is

$$\mathbb{E}[\text{UNIFORM}(X^s)] = \frac{1}{k} \sum_{t=1}^k \mathbb{E}[A_t(X^s)].$$

For $t = s$, we have $\mathbb{E}[A_s(X^s)] \geq \frac{1}{2} \mathbb{E}[\max(X^s)]$, while for $t \neq s$, the reward of A_t on X^s is non-negative. Therefore,

$$\mathbb{E}[\text{UNIFORM}(X^s)] \geq \frac{1}{k} \cdot \frac{1}{2} \mathbb{E}[\max(X^s)] = \frac{1}{2k} \mathbb{E}[\max(X^s)].$$

Thus

$$\min_{s \in [k]} \frac{\mathbb{E}[\text{UNIFORM}(X^s)]}{\mathbb{E}[\max(X^s)]} \geq \frac{1}{2k},$$

and the bound on the competitive ratio follows. ■

While UNIFORM achieves the optimal worst-case guarantee of $1/(2k)$, it does so without using any information from the observed values. This lack of adaptivity limits its ability to improve on easier instances than the worst-case one, where learning from observed outcomes could lead to better decisions, from a practical standpoint.

In the next section, we aim to design more refined algorithms that incorporate learning and adaptivity. Our goal is to match the worst-case guarantee of $1/(2k)$ while improving the empirical performance by taking into account the observed values, and adjusting the decision-making process accordingly.

4.5 Observation-Based Strategies

We now turn to strategies that, in contrast to UNIFORM, use the information revealed during their execution. Our strategies are guided by two central design considerations:

- **Commitment:** Allocating a fixed minimum probability to each scenario in order to ensure the worst-case guarantee. When committed, the algorithm applies the HK strategy for the chosen scenario without adapting.
- **Belief Update:** Using the observed values to dynamically estimate which scenario is most likely, based on Bayesian inference [Gel+13]. This guides the decision-making process when the algorithm has not yet committed.

We discuss these two guiding rules in the next two sections.

4.5.1 Commitment

For the worst-case instance in the proof of Theorem 4.2, achieving a competitive ratio of $1/(2k)$ required committing with equal probability $1/k$ to each scenario in advance. However, this uniform commitment is not always necessary. In general, a lower commitment probability may suffice, freeing some probability mass that can be allocated more judiciously.

We describe a general commitment strategy that guarantees a competitive ratio of $1/(2k)$. For each scenario $s \in \{1, \dots, k\}$, define the *minimum required commitment probability* as:

$$\bar{p}_s = \frac{\mathbb{E}[\max(X^s)]}{2k \cdot \mathbb{E}[\text{HK}(X^s)]}.$$

Consider then a class of algorithms based on the following rule:

- With probability $\bar{p}_s$, commit to scenario s and follow the HK strategy for that scenario throughout the entire process.
- With the remaining probability mass $q = 1 - \sum_{s=1}^k \bar{p}_s$, execute a strategy that adapts based on observed values, as discussed in the next section.

The following theorem establishes the competitiveness of algorithms in this class.

► **Theorem 4.5.** The commitment strategy described above guarantees a competitive ratio of at least $1/(2k)$. ◀

Proof. For any scenario s , the algorithm commits to scenario s with probability $\bar{p}_s$, and when it does so, it applies the HK strategy. By definition of $\bar{p}_s$ and optimality of HK for known distributions, we have:

$$\bar{p}_s \cdot \mathbb{E}[\text{HK}(X^s)] = \frac{1}{2k} \cdot \mathbb{E}[\max(X^s)].$$

The expected reward in scenario s is at least this quantity, regardless of what happens in the non-committed part. Therefore, for all s , the algorithm satisfies:

$$\frac{\mathbb{E}[A(X^s)]}{\mathbb{E}[\max(X^s)]} \geq \frac{1}{2k}.$$

■

This commitment mechanism provides a simple way to ensure the worst-case guarantee while reserving part of the probability mass towards strategies that make use of observed values. We explore such strategies in the next subsection.

4.5.2 Belief Update

We discuss different approaches for allocating probability beyond the commitment part. Recall that we denote by $q = 1 - \sum_{s=1}^k \bar{p}_s$ the probability, which will be allocated to adaptive strategies that use information observed during the process. This is achieved by maintaining a belief distribution over the k scenarios as values are revealed.

Let $b^i = (b_1^i, \dots, b_k^i)$ denote the *belief* vector after observing the first i box values, where b_s^i represents the probability assigned to scenario s based on the values observed up to box i . Initially, i.e., before any values are observed, the belief is uniform: $b_s^0 = 1/k$ for all $s \in [k]$. As the algorithm observes new values, it updates this belief using Bayes' rule.

Specifically, upon observing value v_i from box i , the belief is updated as follows:

$$b_s^i := \frac{b_s^{i-1} \cdot \mathbb{P}(X_i^s = v_i \mid D_i^s)}{\sum_{s'=1}^k b_{s'}^{i-1} \cdot \mathbb{P}(X_i^{s'} = v_i \mid D_i^{s'})},$$

where $\mathbb{P}(X_i^s = v_i \mid D_i^s)$ is the probability of observing v_i under distribution D_i^s . This update rule follows standard Bayesian inference principles (see, e.g., [Gel+13]).

This Bayesian updating mechanism allows the algorithm to gradually shift probability mass toward the more likely scenarios, as data are observed. The belief vector b^i can then be used to guide decisions, such as adjusting thresholds or prioritizing certain scenarios, in a way that reflects the accumulated information.

In the following section, we describe specific strategies that use this belief update process to improve performance while preserving the $1/(2k)$ worst-case guarantee.

4.6 Belief-Driven Decision Rules

We define three different belief-driven algorithms. We only focus on how the algorithms exploit the belief update, since they all must follow the commitment rule, as described in Section 4.5.1.

Algorithm INDEPENDENT: This algorithm samples a scenario s at each box independently, according to the distribution defined by b_s , and applies the corresponding HK threshold.

Algorithm MAX-BELIEF: At each box, the algorithm selects the scenario s^* with the highest belief value b_{s^*} (breaking ties arbitrarily), and applies the HK threshold for scenario s^* . This choice is made independently for each box, potentially resulting in frequent switches between scenarios as the beliefs evolve.

Algorithm CONSISTENT: This algorithm is randomized, and uses strategy s with probability b_s^t at any moment t . However, instead of making an independent choice at each time step t , it minimizes the expected number of strategy changes over time. Intuitively, the algorithm can be described as follows: it chooses a random value r uniformly from the interval $(0, 1]$ and is completely deterministic from that moment on. At any moment t the algorithm maintains a partition $p^t : (0, 1] \rightarrow \{1, \dots, k\}$ of the unit interval, into k regions, each assigned to a scenario. At time 0, before seeing the first value, the beliefs are uniformly distributed, and the initial partition is defined as $p^0(x) = \lceil xk \rceil$. The invariant is that partition is bound to the belief in the sense that for every s, t , the integral $\int_{x:p^t(x)=s} 1 \, dx$ equals b_s^t . The algorithm chooses the scenario $p^t(r)$, and hence it uses scenario s with probability b_s^t . When the t -th value appears and the belief is updated, the partition is updated as well, while minimizing the changes to the partition. Formally, the partition p^t is constructed so as to minimize the integral $\int_{x:p^t(x) \neq p^{t-1}(x)} 1 \, dx$. Consequently, we set $p^t(x) = p^{t-1}(x)$ for scenario s and x such that $p^{t-1}(x) = s$ and $x \leq \delta_s$, where δ_s is the value in $(0, 1]$ such that the measure $\int_{x:p^t(x)=s, x \leq \delta_s} 1 \, dx$ equals $\min\{b_s^{t-1}, b_s^t\}$. The remaining regions in $(0, 1]$ of measure $\sum_{s=1}^k |b_s^t - b_s^{t-1}|$ are arbitrarily assigned to the scenarios, such that the invariant remains satisfied.

We now discuss the implementation of Algorithm CONSISTENT, which is formally described in the pseudocode Algorithm 5. Instead of an actual partition, the algorithm maintains a *histogram*, in which each scenario s is represented by a bar of

Algorithm 5: CONSISTENT

Input: Update of current belief vector $b^t = (b_1^t, \dots, b_k^t)$
Output: Scenario s chosen by the algorithm

- 1 **Initialization (only once):**
- 2 $b_i \leftarrow \frac{1}{k}$ for all $i \in \{1, \dots, k\}$;
- 3 $s \leftarrow \text{Uniform}(\{1, \dots, k\})$;
- 4 $\ell \leftarrow \text{Uniform}(0, b_s)$;
- 5 **Update on belief vector:** Compute an earth mover transformation from distribution b^{t-1} to b^t ;
- 6 **for** each movement of probability mass δ from scenario u to scenario v **do**
- 7 **if** $a = s$ and $\ell \geq b_u - \delta$ **then**
- 8 $\ell \leftarrow \ell - (b_u - \delta) + b_v$;
- 9 $s \leftarrow v$;
- 10 $b_u \leftarrow b_u - \delta$;
- 11 $b_v \leftarrow b_v + \delta$;
- 12 **return** s

height b_s^t . It also maintains a position $\ell \in [0, b_s^t]$ in the bar of the currently selected scenario. Initially, s is chosen uniformly at random, and ℓ is chosen uniformly in $[0, b_s^t]$. The pair (s, ℓ) represents the value r described in the informal explanation above

Whenever the beliefs are updated, so are the bars of the histogram, following the *earth-mover distance* principle. Since the heights of the bars change in small steps, we drop the superscript t , and denote by b_u the current height of the bar associated to scenario u . After each update we have $b_u = b_u^t$ for every scenario u .

Formally, when some probability mass δ is transferred from a scenario u to a scenario v , a top portion of height δ is chopped off the bar for scenario u and stacked on the bar for scenario v . If $s = u$ and $\ell \geq b_u^t - \delta$, then the position ℓ is affected by this transformation, and the algorithm updates $s = v$ and $\ell := \ell - (b_u^t - \delta) + b_v^t$. See Figure 4.1 for an illustration.

All three algorithms described above have a competitive ratio of $1/(2k)$. This follows from the commitment part, which guarantees that each scenario s is chosen with probability at least $\bar{p}_s$, as defined earlier. By Theorem 4.5, this suffices to ensure the worst-case guarantee. The belief update aims to improve performance on easier instances, but the theoretical guarantee comes from the commitment step.

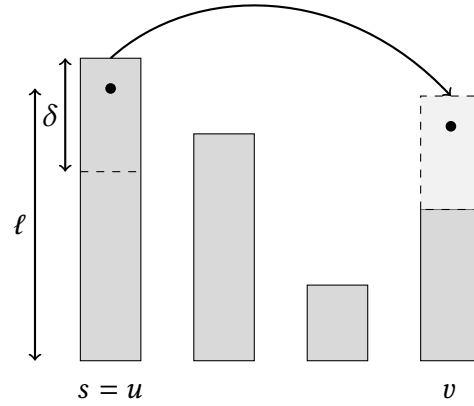


Figure 4.1: Illustration of algorithm CONSISTENT: Here, a mass equal to δ moves from scenario $s = u$ to scenario v .

4.7 Experimental Evaluation

In this section, we evaluate the performance of the proposed algorithms on real-world data. Specifically, we will compare the algorithms UNIFORM, discussed in Section 4.4, and the belief-based algorithms INDEPENDENT, MAX-BELIEF, and CONSISTENT, discussed in Section 4.6. Recall that all algorithm have optimal competitive ratio equal to $\frac{1}{2k}$.

4.7.1 Datasets

The dataset used in our experiments is based on housing market prices in France, sourced from government-provided open data² for a period from 2020 to 2023. For the purposes of our experiments, each month in this time interval represents a “box” and each region in France is treated as a “scenario”. Hence, this setup results in a total of 48 boxes and 13 different scenarios.

The motivation for this experimental setup is a real-world situation in which a homeowner may wish to sell a property, but lacks precise knowledge of how the market will evolve. Homeowners typically have access to historical market data from various regions, which can influence their decision; however, there is uncertainty about which region’s market trends will best represent future prices. This uncertainty is analogous to having several forecasts, each corresponding to market performance in different regions. We simulate this uncertainty by treat-

² <https://www.data.gouv.fr/fr/datasets/5c4ae55a634f4117716d5656/>

ing each region as a potential scenario, where only one scenario will reflect the actual market performance, although the homeowner does not know which one in advance.

After initial preprocessing of the dataset, we eliminated extreme entries, such as properties listed at €1, and those with prices per square meter outside the range of €1,000 to €30,000. This filtration yielded a dataset consisting of 1,103,336 recorded property prices. We then grouped these prices by region, with each scenario containing on average 84,872 sales.

To generate distributions for each month and each region, we first mapped each recorded price to a predefined interval defined on a logarithmic scale. Namely, each price was rounded to the nearest power of 1.1 according to the formula:

$$\text{rounded price} = 1.1^{\lfloor \log_{1.1}(\text{Price}) + 0.5 \rfloor}.$$

Having completed this quantization, we were able to estimate the empirical distribution of house prices within one month by constructing the histogram over these intervals. Figure 4.2 shows an example of these distributions for January 2020 across three regions.

4.7.2 Results

Figure 4.3 depicts the empirical competitive ratio of the various algorithms. To ensure statistical reliability, each reported value is the average of 100,000 simulation runs.

As shown in Figure 4.3, both MAX-BELIEF and CONSISTENT achieve better performance than INDEPENDENT in nearly all regions. For instance, in Auvergne-Rhône-Alpes, INDEPENDENT has empirical competitive ratio of 0.72, while MAX-BELIEF and CONSISTENT achieve ratios 0.85 and 0.84, respectively. Similar improvements are seen in other regions. The Île-de-France region is a notable exception, where all three algorithms achieve similarly high performance, with competitive ratios of 0.89 for INDEPENDENT, 0.89 for MAX-BELIEF, and 0.86 for CONSISTENT. The small difference, in this case, can be explained by the high and relatively stable housing prices in Île-de-France, which reduces the sensitivity to the threshold selection.

To better explain the above finding, Figure 4.4 depicts the thresholds chosen by algorithm HK for each scenario. For the majority of scenarios (i.e., regions), the thresholds are similar, with the exception of Île-de-France region, where housing prices are significantly higher. Hence, if Île-de-France is the real scenario, all algorithms use relatively small thresholds no matter how they use their beliefs, and accept a box with relatively high reward.

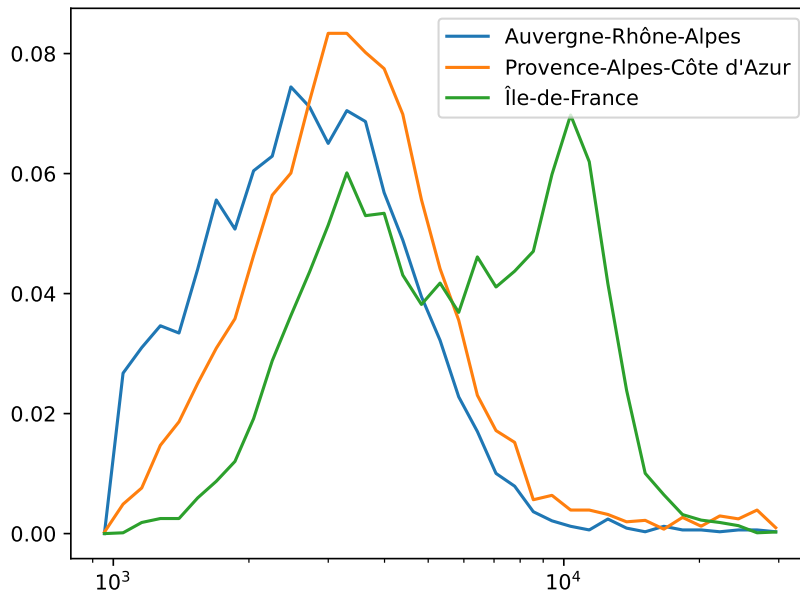


Figure 4.2: Distribution of housing prices per square meter in January 2020 for three regions: Auvergne-Rhône-Alpes, Provence-Alpes-Côte d’Azur, and Île-de-France.

To evaluate more precisely the impact of the belief update phase on the performance of the algorithms, we focused on the three regions with the highest number of property sales: Auvergne-Rhône-Alpes, Provence-Alpes-Côte d’Azur, and Île-de-France. In addition to the previously described algorithms, we introduced a new family of strategies called **LEARN x** , where x denotes the number of initial boxes used only for observing values. During this observation phase, the algorithm collects information, but does not make any decisions. After seeing the x first boxes, it selects the most likely scenario based on the updated beliefs and then applies the **CONSISTENT** procedure described in Algorithm 5. This allows the algorithm to first gather evidence before committing to a fixed decision rule.

Figure 4.5 depicts the empirical competitive ratio of the various strategies, assuming three scenarios. The goal is to assess the benefits of the different learning techniques. We observe that the baseline algorithm **UNIFORM** performs worse than all other algorithms, especially if Île-de-France is the true scenario, and belief-driven methods consistently outperform **UNIFORM**. In Auvergne-Rhône-Alpes, **UNIFORM** achieves a competitive ratio of 0.94, while both **MAX-BELIEF** and **CONSISTENT** im-

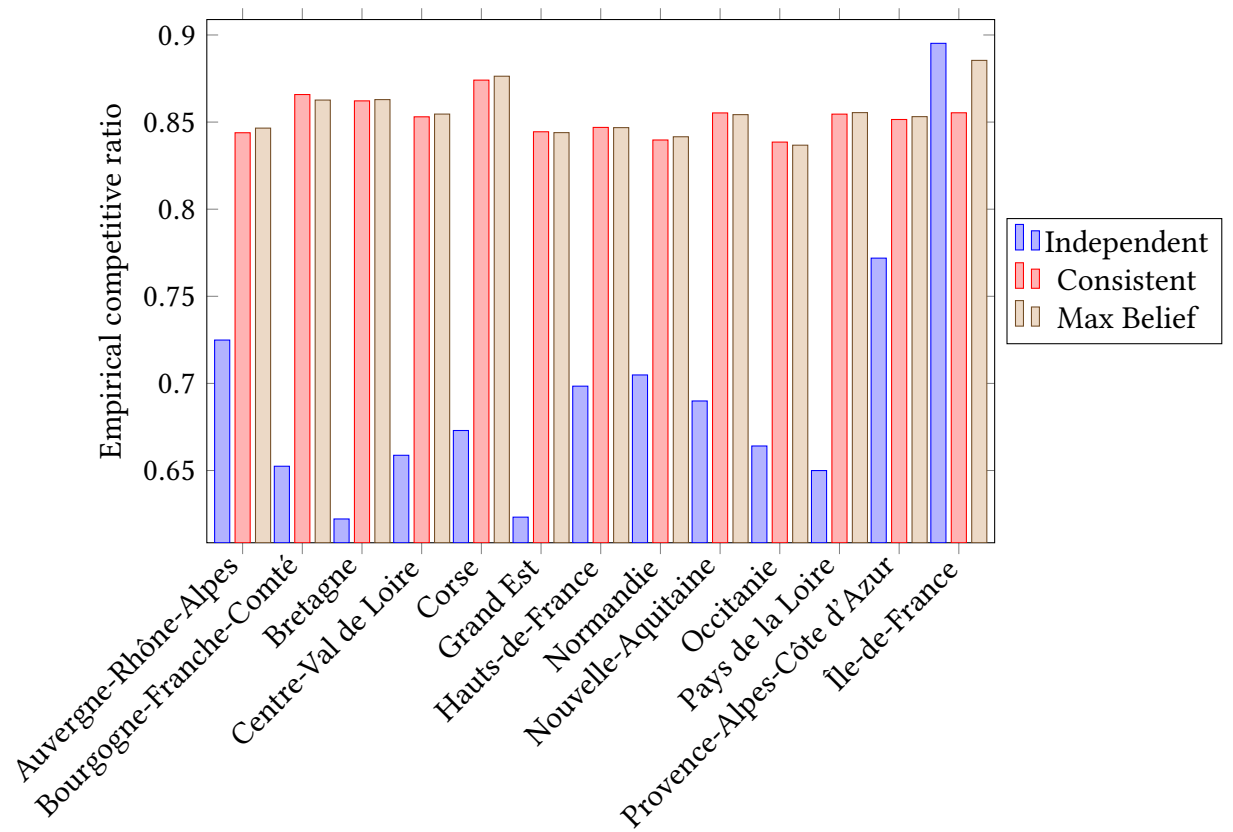


Figure 4.3: Comparison of INDEPENDENT, MAX-BELIEF, and CONSISTENT algorithms. The horizontal axis indicates the true scenario.

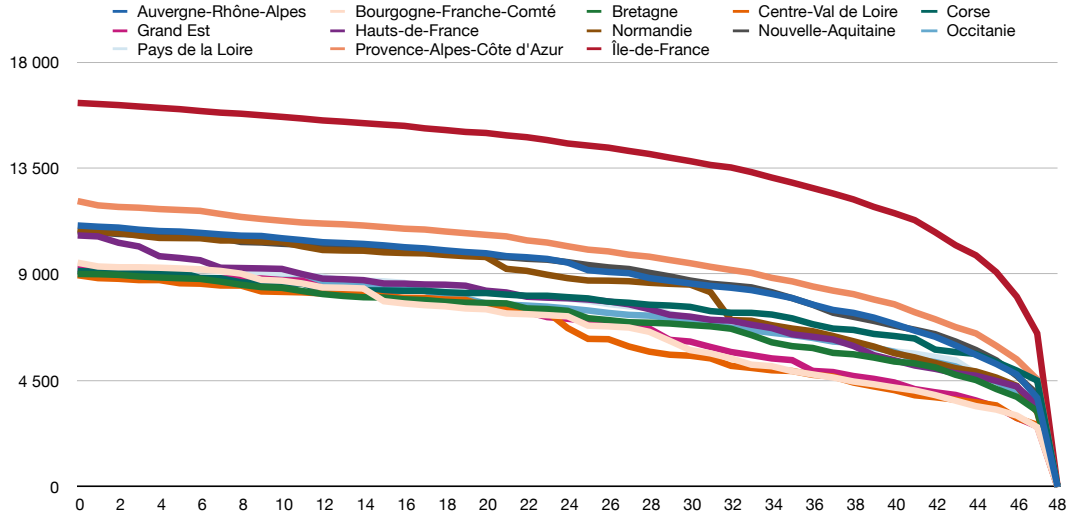


Figure 4.4: Optimal thresholds for each scenario according to the HK algorithm.

prove to 0.97. Among the LEARN x variants, LEARN 5 performs best with a ratio 0.97, while LEARN 30 slightly degrades to 0.95, suggesting that a long observation phase may lead to weaker performance due to delayed commitment. In Provence-Alpes-Côte d’Azur, the empirical competitive ratio increases from 0.96 for UNIFORM to 0.97 for both MAX-BELIEF and CONSISTENT, with LEARN 5 reaching 0.98. In Île-de-France, where housing prices are generally higher, all algorithms perform slightly worse than other regions, in relative terms: UNIFORM achieves ratio 0.88, while both MAX-BELIEF and CONSISTENT reach a ratio 0.93, and the LEARN x variants yield similar values around 0.93 for all tested values of x .

These results show that all algorithms incorporating belief updates and a commitment phase consistently outperform the UNIFORM baseline and demonstrate the advantage of using learning techniques for adaptive decision-making under uncertainty.

4.8 Conclusion

In this chapter, we extended the prophet inequalities problem to a scenario-based setting, where the decision-maker does not know in advance which scenario will materialize. Our main theoretical contribution is a tight upper bound on the competitive ratio that shows that no algorithm can achieve a competitive ratio greater than $1/(2k)$, where k is the number of scenarios. We then gave several algorithms that combine the optimal theoretical guarantee with adaptivity on the

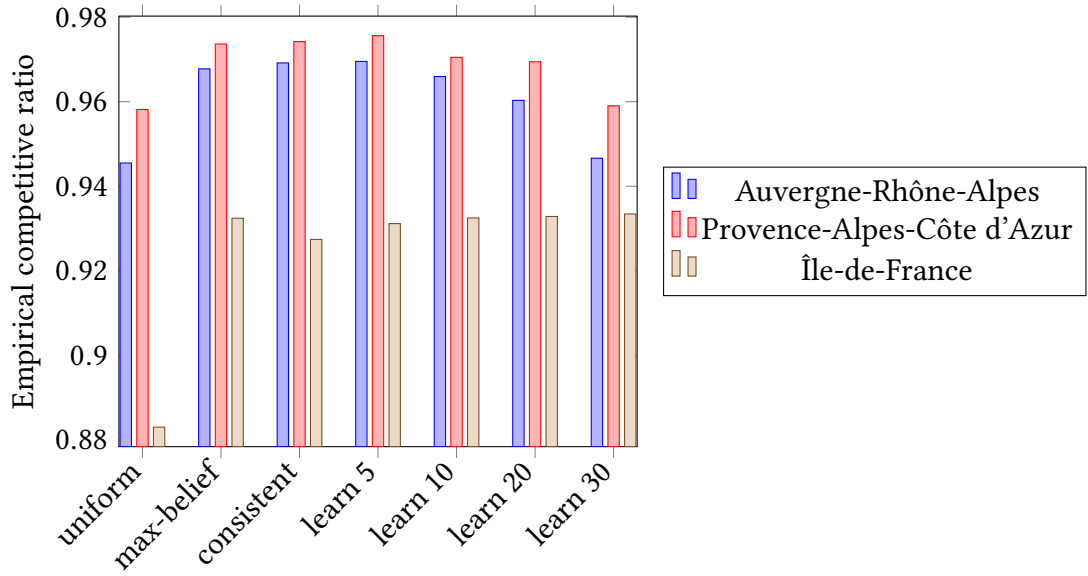


Figure 4.5: Comparison of various algorithms over three possible scenarios.

observed values. Specifically, we showed how to use belief updates so as to improve the performance on typical instances, other than the worst-case one. To evaluate the algorithms, we performed an experimental evaluation using real-world data from the French housing market. The findings demonstrate that belief-based algorithms offer a significant improvement over the baseline (uniform) strategy across a wide range of scenarios.

A natural direction for future work is to investigate the independent and identically distributed (i.i.d.) setting, where values in each scenario are drawn independently from the same underlying distribution. For the original prophet inequalities problem with a single scenario, the i.i.d. case was introduced in [HK82] and has been well-studied, e.g., [Cor+17; Ker86]. The known tight bound on the competitive ratio is equal to, and its proof relies on sophisticated arguments [Cor+17]. Extending these ideas to the multiple-scenario setting poses new challenges, as it becomes necessary to understand how the scenario uncertainty interacts with the probabilistic structure of the input. An additional difficulty lies in combining belief updates across scenarios with the stochastic behavior of value sequences, which complicates both the algorithm design and the analysis.

5 Decision-Theoretic Approaches in Learning-Augmented Algorithms

5.1 Introduction

In this chapter we study learning-augmented algorithms for a decision-theoretic point of view. We refer the reader to Section 2.1.2 for some context, and the definitions of concepts such as the consistency, the robustness, and the performance ratio.

Our starting point is that in many recent works, learning-augmented algorithms are evaluated using two main objectives: the consistency, which measures performance under perfect predictions, and the robustness, which captures performance under adversarial prediction errors. A typical goal is to find Pareto-optimal algorithms that offer the best possible trade-off between these two measures. However, such algorithms may behave poorly in practice, especially when the prediction error is small but not zero. This issue, known as brittleness, has been observed in several learning-augmented problems [Ele+24]. Even when brittleness is avoided, one is often left with an entire class of Pareto-optimal algorithms and it is not immediately obvious how to select the “best” one [BP25].

An alternative, and much different approach is to enforce smoothness, which ensures that performance degrades gradually as the prediction error increases. This is often done by assuming a known upper bound on the prediction error, sometimes referred to as a *confidence parameter*. The algorithm is then designed and analyzed with respect to this bound [AKZ22; Ant+23]. While this approach may avoid brittleness, it may result in algorithms that are suboptimal when the prediction is highly accurate, since they are not designed to take advantage of such situations. Moreover, it requires knowledge or estimation of the confidence parameter, which may not always be available.

Motivated by these limitations, in this chapter we propose a more global approach. Rather than focusing only on specific ranges (or values) of prediction accuracy, we aim to evaluate and compare algorithms based on how they perform across the entire range of possible errors. This naturally motivates the use of techniques from *decision theory*, an area that addresses precisely the issue of choosing of the most suitable solution among many possible alternatives.

5.1.1 Contributions

We study how to apply decision-theoretic methods so as to compare learning-augmented algorithms. The goal is to select the algorithm that performs “best overall”, based on its performance function and using clear, measurable criteria. We introduce two types of methods: deterministic ones, which do not rely on any assumptions about how accurate the prediction is, and stochastic ones, which take into account the uncertainty of the prediction and allow us to reason about risk. In particular, we focus on the following measures:

Distance measures. Here, we evaluate the *distance* between the performance of the algorithm, and an *ideal* solution, i.e. an omniscient algorithm that knows the input, but is constrained by the same robustness requirement as the online algorithm. We focus on two distance metrics: i) The *weighted* maximum distance, which is defined as the weighted L_∞ -norm distance between the performance function of the algorithm and that of the ideal solution; here, the weight is a user-specified function that reflects how much, and what type of importance the designer assigns to prediction errors; and ii) The *average* distance, which measures the aggregate distance between the algorithm and the ideal solution, averaged over the range of the prediction error.

Distance measures are inspired by tools such as Receiver Operating Characteristic (ROC) graphs [Faw06], which have long been used to describe the tradeoff between the true positive rates (TPR) and the false positive rates (FPR) of classifiers. Distance metrics between two ROC curves have been used as a comparison measure of classifiers. Moreover, weighted distances in ROC graphs can help emphasize critical regions: e.g., a user who is sensitive to false positives when FPR is low. This weighted approach has several applications in medical diagnostic systems [JJ10].

Risk measures. Here, the motivation comes from the realization that Pareto-optimal algorithms and algorithms with confidence parameters are designed around a notion of *risk*: namely, the risk due to deviating from a perfect prediction. This observation helps explain some of their undesirable characteristics such as their brittleness and inefficiency. We first formalize the notion of risk by introducing a stochastic prediction setting that provides imperfect distributional information to the algorithm. We then introduce a novel analysis approach based on a risk measure that has been influential in decisions sciences, namely the *conditional value-at-risk*, denoted by CVaR_α , which informally measures the expectation of a random loss/reward on its $(1 - \alpha)$ -fraction of worst outcomes [SSU08]. Here, $\alpha \in [0, 1)$, is a parameter that measures the *risk aversion* of the end user. We show how to obtain a parameterized analysis based on risk-aversion, which allows us to

quantify the trade-off between the performance of the algorithm and its risk due to the error.

Our techniques generalize previous known approaches in learning-augmented algorithms. More precisely, in the context of distance measures, we show that the appropriate choice of the weight function can help recover both the Pareto-optimal and those with a confidence parameter, later referred to as δ -tolerant. The same holds for the risk-based analysis; here we obtain a generalization of the *distributional* consistency-robustness tradeoffs of [Dia+21], by introducing the notion of α -consistency, where α is the risk parameter.

The remainder of the chapter is structured as follows. In Section 5.2 we formally describe the decision-theoretic framework of our study. We then apply the various distance and risk measures to various problems. For ski rental (Section 5.3) we show how to find among the infinitely many algorithms the one that optimizes our metrics. For 1-max search (Section 5.4) we show how to find, for any parameter r , an algorithm that likewise optimizes the metrics, among the infinitely many r -robust strategies. Last, for contract scheduling (Section 5.5), we show how to find, among the infinitely many schedules of optimal acceleration ratio, one that simultaneously optimizes the robustness as well as each of our target metrics. Due to space limitations, this study is presented in the appendix. In Section 5.6, we provide an experimental evaluation of our algorithms that demonstrates the performance improvement that can be attained relative to the state-of-the-art algorithms.

5.1.2 Other related work

We discuss some relevant results and works, additionally to those in Section 2.1.3.

Ski-rental, 1-max search and contract scheduling

In the ski rental problem, the goal is to decide when to stop renting and buy, given a prediction on the total duration. The learning-augmented version of the continuous setting has been studied extensively [Ang+24b; KPS24; WZ20], with several families of algorithms proposed to balance robustness and consistency. Among these, Pareto-optimal approaches offer different trade-offs depending on the error. A recent extension [BP25] introduces a parameterized family with smoother behavior as the prediction varies. These algorithms will be discussed in detail later.

For 1-max search, predictions on the maximum value can help improve over worst-case threshold strategies. Several algorithms have been proposed, including Pareto-optimal ones [Sun+21] and smooth alternatives based on a confidence parameter [AKZ22]. However, recent work [Ele+24] has shown that Pareto-optimal

strategies may be brittle when the prediction is slightly inaccurate. We return to this issue in the corresponding section and present comparisons between these approaches and our methods.

In contract scheduling, a prediction on the interruption time can be used to improve the acceleration ratio. Algorithms proposed in [AK23a] achieve Pareto-optimality under robustness and consistency constraints, but also suffer from brittleness in the presence of small prediction errors [Ele+24]. We analyze and compare these algorithms further in the chapter.

Decision theory and algorithm evaluation

Decision-theoretic tools have been applied in both online algorithms and combinatorial optimization. In online algorithms, one important class of problems is sequential decision-making, where an algorithm repeatedly selects actions and receives feedback in the form of rewards or losses. A typical example is the *multi-armed bandit problem*, where the goal is to minimize the total loss over time. In these settings, various *utility-based* and *risk-sensitive* criteria have been used to guide decision-making under uncertainty. To better evaluate the performance when the environment changes over time, refined notions such as *policy regret* have been introduced [DTA12; MLS22]. Unlike standard regret, which compares to the best fixed action, policy regret compares to the best sequence of actions, allowing a more realistic assessment in dynamic conditions.

In combinatorial optimization, decision-theoretic ideas have been used to handle problems with uncertain inputs. One approach is to allow small adjustments to the computed solution after some uncertainty is resolved, a method known as *incremental recourse* [NO13]. Another approach models the uncertainty through a finite set of scenarios and formulates the problem as *multicriteria optimization*, where each scenario defines one criterion. For example, in [PSS06], solutions are compared using *cost vectors* and evaluated through tools such as *Lorenz dominance* and *Ordered Weighted Averaging (OWA)* operators. These methods provide a principled way to select solutions that perform well across multiple possible outcomes.

The conditional value-at-risk (CVaR) was recently used in [Chr+24] to design and analyze *randomized* algorithms in standard settings without predictions. However, to the best of our knowledge, CVaR has not been studied in the context of competitive analysis for learning-augmented algorithms. For a related discussion and a comparison with scenario-based robust optimization, especially in applications such as financial decision-making, see Section 2.2.

5.2 Decision-theoretic models

In this section, we formalize the key notions that we introduce and apply in this work. These definitions apply to general cost minimization problems, such as ski rental, but can be naturally extended to profit maximization problems such as 1-max search and contract scheduling. We denote by $\text{OPT}(\sigma)$ the cost of an optimal offline algorithm on an input sequence σ .

The basic setup and motivation for learning-augmented algorithms, including the notions of consistency, robustness, and smoothness, were introduced in Section 2.1.2. We now build on that framework to define more precisely the models and metrics we will use for algorithmic evaluation.

5.2.1 Distance-based analysis

We focus on settings in which the algorithm receives a single-valued prediction. Let σ denote the input, and let x_σ be the associated target quantity (e.g., maximum value in 1-max search or duration in ski rental). The algorithm also receives a prediction $y \in \mathbb{R}$, and the prediction error is defined as $\eta = |x_\sigma - y|$.

We define the *range* of a prediction y as an interval $R_y = [\ell, u] \subseteq [0, \infty)$ such that $x_\sigma \in R_y$. This formulation allows us to study algorithms with a tolerance parameter. In particular, if $R_y = [(1 - \delta)y, (1 + \delta)y]$ with $\delta \in [0, 1]$, we refer to δ -tolerant algorithms. We emphasize that a bounded-error assumption is not necessary in our framework; unless specified, we consider the general case $R_y = [0, \infty)$ and use the δ -tolerant form for comparison with known results.

Let A be an online algorithm for a cost minimization problem. Given input σ and prediction y , let $A(\sigma, y)$ denote the cost incurred by A . We define the performance ratio as:

$$\text{perf}(A, \sigma, y) = \frac{A(\sigma, y)}{\text{OPT}(\sigma)}.$$

For profit maximization problems, we instead use $\text{OPT}(\sigma)/A(\sigma, y)$; the definitions and analysis are otherwise analogous.

In order to define our distance measures, we first introduce the concept of an *ideal* solution. Given a robustness requirement r and an input σ , we define $I_r(\sigma)$ as the *smallest cost* that can be achieved on σ by an online algorithm that is required to be r -competitive on all inputs. We then set $\text{perf}(I_r, \sigma) = \frac{I_r(\sigma)}{\text{OPT}(\sigma)}$. By definition, any r -robust online algorithm A with prediction y obeys $\text{perf}(A, \sigma, y) \geq \text{perf}(I_r, \sigma)$.

With this ideal benchmark in place, we define our distance measures. We begin with the *maximum* weighted distance. The user specifies a *weight* function $w_y :$

$R_y \rightarrow [0, 1]$, which quantifies the importance assigned to prediction errors and aims to guarantee smoothness. We require that w_y is piecewise monotone: it is non-decreasing on $[\ell, y]$ and non-increasing on $[y, u]$. The maximum distance of an r -robust algorithm A , given prediction y , is

$$d_{\max}(A) = \sup_{\sigma, x \in R_y} \{(\text{perf}(A, \sigma, x) - \text{perf}(I_r, \sigma))w_y(x)\}. \quad (5.1)$$

Thus, the maximum distance measures the weighted worst-case deviation from the ideal performance. Note that if w satisfies $w(y) = 1$ and $w(x) = 0$ for all $x \neq y$, minimizing $d_{\max}$ reduces to finding a Pareto-optimal algorithm.

We also define the *average* weighted distance, which measures the average deviation from the ideal performance across the prediction-error range:

$$d_{\text{avg}}(A) = \sup_{\sigma} \frac{1}{|R_y|} \int_{R_y} (\text{perf}(A, \sigma, z) - \text{perf}(I_r, \sigma))w_y(z) dz. \quad (5.2)$$

5.2.2 Risk-based analysis

Since risk is an inherently stochastic concept, we need to introduce stochasticity in the prediction model. More precisely, we will assume that the prediction is in the form a distribution μ , with support over an interval $[\ell, u] \in \mathbb{R}$, and a pdf that is non-decreasing in the interval $[\ell, y]$ and non-increasing in $[y, u]$. This model has two possible interpretations. First, one may think of μ as a *distributional* prediction, in the lines of stochastic prediction oracles [Dia+21]. A second, but related interpretation of μ is that of a *prior* on the predicted value, based on historical data. We will use R_μ to refer to the range of μ , since it is motivated by considerations similar to the notion of range in the distance measures.

Our analysis will rely on the Conditional Value-at Risk (CVaR) measure from Financial Mathematics [RU+00; RU02]. Considering a cost-minimization problem, for a random variable X , and a parameter $\alpha \in [0, 1)$ that describes the risk *aversion*, CVaR_α is defined as

$$\text{CVaR}_\alpha(X) = \inf_t \left\{ t + \frac{1}{1 - \alpha} \mathbb{E}[(X - t)^+] \right\}, \quad (5.3)$$

where $(X - t)^+ = \max\{X - t, 0\}$.

Let $\mathcal{F}$ denote the class of *input* distributions (i.e., distributions over sequences σ) in which the predicted information has the same distribution as μ . For example, in 1-max search, F is a distribution of input sequences such that the maximum price

is distributed according to μ . For an algorithm A , and given $\alpha \in [0, 1)$, we define the α -consistency of A as

$$\alpha\text{-cons}(A) = \sup_{F \in \mathcal{F}} \left(\frac{\text{CVaR}_{\alpha, F}(A(\sigma))}{\mathbb{E}_{\sigma \sim F}[\text{OPT}(\sigma)]} \right), \quad (5.4)$$

where the subscript F in the notation of CVaR signifies that σ is generated according to F .

We can then summarize our objective as follows. Given a robustness requirement r , and a risk parameter α , we would like to find an r -robust algorithm of minimum α -consistency.

This measure is a risk-inclusive generalization of consistency, and interpolates between two extreme cases. The first case, when $\alpha = 0$, describes a *risk-seeking* algorithm that aims to maximize its expected profit without considering deviations from the distributional prediction. In this case, $\text{CVaR}_{\alpha, F}(A) = \mathbb{E}_{\sigma \sim F}[A(\sigma)]$, thus (5.4) is equivalent to the consistency of A in the distributional prediction model of [Dia+21]. The second case, when $\alpha \rightarrow 1$, describes a *risk averse* algorithm: here, it follows that $\text{CVaR}_{\alpha, F}(A) = \inf_{\sigma \in \text{supp}(F)} A(\sigma)$, thus (5.4) describes the performance of A in the adversarial situation in which all the probability mass is located to a worst-case point within the prediction range. Note that this risk-based model is an adaptation of risk-sensitive randomized algorithms [Chr+24] to learning-augmented settings.

5.3 Application in ski rental

We consider the continuous version of the ski rental problem, in contrast to the discrete version presented in Section 2.1.3. In this setting, skis can be bought at any real-valued time. Let b denote the buying cost, and x the (unknown) number of skiing days. For an algorithm that buys at time T , the total cost is x if $x < T$, and $b + T$ if $x \geq T$. The best deterministic competitive ratio is equal to 2.

In the learning-augmented setting, the algorithm receives a prediction y of the number of skiing days. It is easy to verify that for any $r \geq 2$, an r -robust algorithm must purchase within the interval $[b/(r-1), b(r-1)]$. We denote by A_T , where $T \in [b/(r-1), b(r-1)]$, the r -robust algorithm that buys at time T . It was shown in [KPS24; WZ20] that all algorithms of this form have consistency at most $r/(r-1)$. More generally, the class of algorithms proposed in [BP25] achieves a Pareto-optimal trade-off between consistency and robustness. Specifically, it buys skis at time $b/(r-1)$ when $y \geq b$, and at time $\rho \in [b, b(r-1)]$ when $y < b$. The

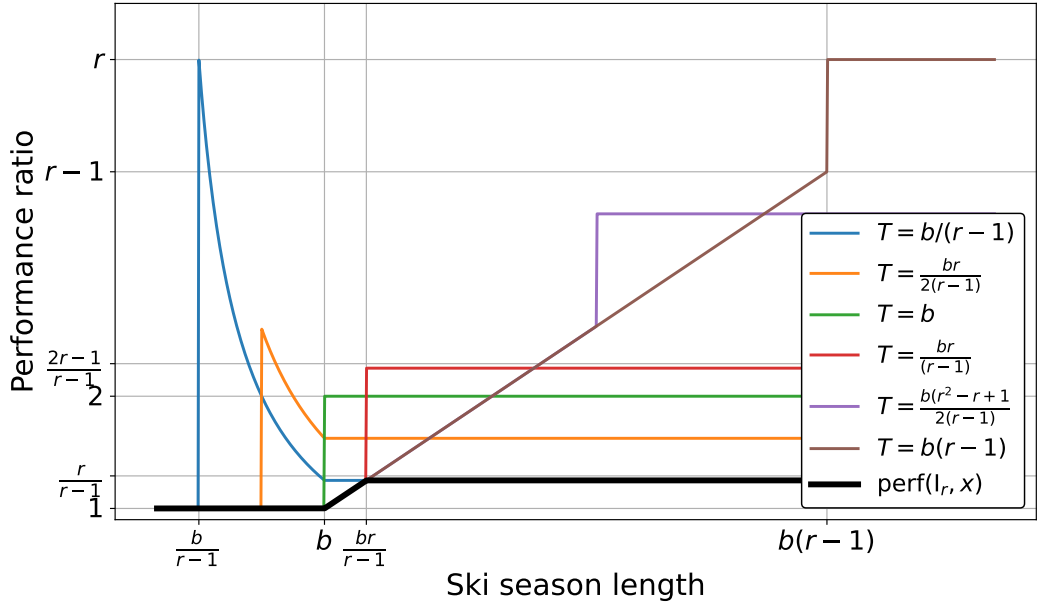


Figure 5.1: Performance of the ideal r -robust algorithm I_r compared to several algorithms A_T for different values of T .

parameter ρ controls the smoothness of the algorithm's performance, with each choice yielding a distinct trade-off curve.

Objective: Given a robustness requirement r and a prediction y on the number of skiing days, find the optimal $T \in [b/(r-1), b(r-1)]$ such that A_T minimizes the distance-based objective defined in Section 5.2.

We will denote by $T_{\max}^*$, T_{avg}^* and T_{cvar}^* the optimal values according to the maximum/average distance, and according to CVaR, respectively.

5.3.1 Distance Measures

We begin by expressing the performance of the ideal r -robust algorithm I_r .

► **Lemma 5.1.** Given a robustness requirement r , the performance ratio of the ideal algorithm I_r is

$$\text{Perf}(I_r, x) = \begin{cases} 1, & \text{if } x < b \\ x/b, & \text{if } x \in [b, \min\{\frac{br}{r-1}, b(r-1)\}] \\ \frac{r}{r-1}, & \text{if } x \geq \min\{\frac{br}{r-1}, b(r-1)\}. \end{cases} \quad (5.5)$$



Proof. First, recall that in order to be r -robust, any algorithm must buy skis no later than time $b(r-1)$ and no earlier than $b/(r-1)$. We consider the following possible cases.

Case 1: $x < b$.

In this case, I_r rents up to time b . This guarantees r -robustness, and since $I_r(x) = x$, we have that $\text{perf}(I_r, x) = 1$.

Case 2: $x \in [b, \min\{\frac{br}{r-1}, b(r-1)\}]$.

In this case, I_r buys at time $\min\{\frac{br}{r-1}, b(r-1)\} > x$, so $I_r(x) = x$. This guarantees r -robustness, since the buy time lies in $[b/(r-1), b(r-1)]$.

Now consider any r -robust algorithm A_T , then we consider two possible cases on T .

- If $T < b$, then A_T buys before b , and its cost is at least that of $A_{b/(r-1)}$, which is

$$\text{cost}(A_{b/(r-1)}, x) = \frac{b}{r-1} + b = \frac{br}{r-1} > x.$$

- If $T \in [b, \min\{\frac{br}{r-1}, b(r-1)\}]$, then $\text{cost}(A_T, x) = T + b \geq x$.

In both cases, $\text{cost}(A_T, x) \geq I_r(x) = x$. Therefore, $\text{Perf}(I_r, x) = x/b$ is optimal among r -robust algorithms.

Case 3: $x \geq \min\{\frac{br}{r-1}, b(r-1)\}$.

In this case, I_r buys at time $b/(r-1)$, which satisfies r -robustness and its cost is

$$I_r(x) = \frac{br}{r-1},$$

while the offline optimum is b .

Now consider any other r -robust algorithm A_T . If $T > b/(r-1)$, then $\text{cost}(A_T, x) = T + b > \frac{br}{r-1} = I_r(x)$. If $T < b/(r-1)$, then A_T is not r -robust.

Therefore, $I_r(x)$ has the minimum cost among all r -robust algorithms, and

$$\text{Perf}(I_r, x) = \frac{br}{(r-1)b} = \frac{r}{r-1}.$$

This completes the proof. ■

For some intuition on the proof, we distinguished between three cases. If $x < b$, then I_r buys at b . If $x > b$, it buys at $\min\{\frac{br}{r-1}, b(r-1)\}$; and if $x \geq$

$\min\{\frac{br}{r-1}, b(r-1)\}$ it buys at $b/(r-1)$. These choices optimize its cost on input x , while guaranteeing r -robustness on all inputs. Note that $\text{perf}(I_r, x)$ has a discontinuity at $x = (r-1)b$ only if $\frac{r}{r-1} \geq r-1$, or $r < 2.618$, approximately.

► **Remark 5.2.** For simplicity, we will focus on the case $r > 2.618$, for which the ideal performance function is continuous. The case $r \in [2, 2.618]$ can be treated analogously, as all the underlying intuition and calculations remain the same. ◀

Consider now any r -robust online algorithm A_T , hence $T \in [b/(r-1), b(r-1)]$. We have that

$$\text{cost}(A_T, x) = \begin{cases} x, & \text{if } x < T, \\ T + b, & \text{if } x \geq T. \end{cases} \quad (5.6)$$

Figure 5.1 illustrates the performance of the ideal algorithm (in black, bold line) and various algorithms in the class A_T . Note that all algorithms have worse performance than the ideal on all inputs, as expected. Note also that no online algorithm dominates the others.

We now move to the computation of the distance measures. We first observe that we can distinguish between algorithms that buy at times $[b/(r-1), b)$, and those that buy at times $[b, b(r-1)]$; we denote the first class by $C_{<b}$ and $C_{\geq b}$, respectively.

From (5.1), and a prediction y with range R_y we have that

$$d_{\max}(A_T) = \sup_{x \in R_y} \left(\frac{\text{cost}(A_T, x)}{\min\{x, b\}} - \text{perf}(I_r, x) \right) \cdot w(x). \quad (5.7)$$

To gain some insight into the structure of the maximum distance objective in (5.7), let us first consider the unweighted case, i.e., $w(x) = 1$. If R_y is unbounded, then $d_{\max}^* = 1$, which is attained by all $A_T \in C_{\geq b}$. Note that among these algorithms, A_b has the best consistency, so we may choose this algorithm as a tie-breaker.

That is, if we have no confidence on the quality of the prediction, the best algorithm is the competitively optimal one, which agrees with intuition. If, however, R_y has known bounds, then the best algorithm depends on this range. For instance, if the right endpoint u of R_y is smaller than b , then any algorithm in the class $C_{\geq b}$ is optimal according to $d_{\max}$. This is consistent with all algorithms in the class defined by [BP25]. It is also interesting to note that regardless of the range R_y , the best algorithm according to $d_{\max}$ is either a single algorithm in $C_{<b}$, namely A_b , or a choice of algorithms in $C_{\geq b}$; this is precisely the set of all algorithms in [BP25]. The following theorem shows how to obtain the best threshold for the weighted distance.

► **Theorem 5.3.** There is an algorithm for computing $T_{\max}^*$ that runs in time linear in the number of critical points of the weight function w in the range R_y . ◀

Proof. For general weight functions $w(x)$, we assume w is symmetric and piecewise monotone: non-decreasing for $x < y$ and non-increasing for $x > y$. The behavior of the maximum distance objective in (5.7) depends on the location of y relative to b , and on the buying threshold T .

By Lemma 5.1, $\text{perf}(I_r, x) = 1$ for $x < b$ and is increasing for $x \geq b$. Thus, any $T \in C_{<b}$ incurs strictly positive distance on $x < b$, whereas algorithms from $C_{\geq b}$ match the ideal on $x < b$ and may be preferable when the weight mass is concentrated around $y < b$.

Given this structure, we partition the computation into two subproblems: (i) optimize over $C_{<b} = [\frac{b}{r-1}, b)$ to obtain T_1^* , and (ii) optimize over $C_{\geq b} = [b, b(r-1)]$ to obtain T_2^* . The two classes lead to different distance expressions because the denominator in (5.7) is $\min\{x, b\}$, i.e., x on $x < b$ and b on $x \geq b$. We then choose

$$T_{\max}^* = \arg \min_{T \in \{T_1^*, T_2^*\}} d_{\max}(A_T).$$

It suffices to evaluate a finite set of *critical points* in R_y where the weighted distance may attain its maximum. These include y , b , the values $\frac{br}{r-1}$ and $b(r-1)$, and any $x \in R_y$ where the derivative of the corresponding weighted distance expression is zero. The resulting procedure in Algorithm 6 runs in time linear in the number of critical points. ■

We now turn to the average distance objective, which from (5.2) is equal to

$$d_{\text{avg}}(A_T) = \frac{1}{|R_y|} \int_{R_y} \left(\frac{A_T(z, y)}{\min\{z, b\}} - \text{perf}(I_r, z) \right) w(z) dz. \quad (5.8)$$

In the unweighted case ($w = 1$), and without assumptions on R_y , $A_{b(r-1)}$ minimizes the average distance. This is because, for $x \geq \frac{br}{r-1}$, its performance ratio matches the ideal, as depicted by the blue curve in Figure 5.1. When R_y is bounded, the optimal algorithm depends on the range, and the problem reduces to minimizing the area between the performance curves of A_T and I_r over R_y . For general weight functions, the integral in (5.8) is evaluated piecewise, depending on w and T .

Algorithm 6: Algorithm for computing $T_{\max}^*$ **Input:** Prediction y with range R_y , weight $w(x)$, robustness r , buy cost b **Output:** Optimal buy time $T_{\max}^*$ minimizing max distance1 **Case 1:** $T \in C_{<b}$;

2 Define critical set

$$S_1 \leftarrow \{y, b\} \cup \left\{ z \in R_y : \frac{d}{dz} \left(\left(\frac{A_T(z)}{z} - 1 \right) w(z) \right) = 0 \right\}.$$

foreach $T \in S_1$ **do**3 Compute $d_{\max}(A_T) \leftarrow \max_{x \in R_y} \left(\frac{\text{cost}(A_T, x)}{x} - 1 \right) w(x)$;4 $T_1^* \leftarrow \arg \min_{T \in S_1} d_{\max}(A_T)$;5 **Case 2:** $T \in C_{\geq b}$;

6 Define critical set

$$S_2 \leftarrow \left\{ y, b, \frac{br}{r-1}, b(r-1) \right\} \cup \left\{ z \in R_y : \frac{d}{dz} \left(\left(\frac{A_T(z)}{b} - \text{perf}(I_r, z) \right) w(z) \right) = 0 \right\}.$$

foreach $T \in S_2$ **do**7 Compute $d_{\max}(A_T) \leftarrow \max_{x \in R_y} \left(\frac{\text{cost}(A_T, x)}{b} - \text{perf}(I_r, x) \right) w(x)$;8 $T_2^* \leftarrow \arg \min_{T \in S_2} d_{\max}(A_T)$;9 **return** $T_{\max}^* \leftarrow \arg \min \{ d_{\max}(A_{T_1^*}), d_{\max}(A_{T_2^*}) \}$;**5.3.2 Risk-Based Analysis**

We consider the CVaR-based analysis of ski rental. In this setting, the algorithm has access to a distributional prediction μ over the skiing horizon x . Following the discussion in Section 5.2.2, we will evaluate an online algorithm A_T by means of its α -consistency (5.4).

Define

$$q_T = \mathbb{P}[A_T(x) = T + b] = \int_T^\infty \mu(x) dx.$$

With this definition in place, we obtain the following result.

► **Theorem 5.4.** Let $x \sim \mu$ and t^* denote the α -quantile of μ , i.e., the value satisfying $\int_0^{t^*} \mu(z) dz = \alpha$. Then

$$\text{CVaR}_{\alpha, \mu}[A_T(x)] = \min \left\{ \frac{1}{1-\alpha} \left(\int_{t^*}^T z \mu(z) dz + (T+b) q_T \right), T + b \frac{q_T}{1-\alpha}, T + b \right\}, \quad (5.9)$$

and

$$T_{\text{CVaR}}^* = \arg \min_{T \in \left[\frac{b}{r-1}, b(r-1)\right]} \text{CVaR}_{\alpha, \mu}[A_T(x)].$$



Proof of Theorem 5.4. Fix $T \in \left[\frac{b}{r-1}, b(r-1)\right]$ and let $x \sim \mu$ denote the predicted skiing horizon. Let $M(t) := \int_0^t \mu(z) dz$ be the cumulative distribution function, and $q_T = 1 - M(T)$ be the probability of the horizon being at least T . The cost of algorithm A_T is

$$A_T(x) = \begin{cases} x, & x < T, \\ T + b, & x \geq T. \end{cases}$$

We evaluate $\text{CVaR}_{\alpha, \mu}[A_T(x)]$ by considering three possible ranges of t in the definition above.

Case 1: $t < T$. In this case,

$$(A_T(x) - t)^+ = \begin{cases} x - t, & t \leq x < T, \\ T + b - t, & x \geq T, \\ 0, & x < t. \end{cases}$$

Multiplying the CVaR expression by $(1 - \alpha)$ gives

$$\begin{aligned} (1 - \alpha) \text{CVaR}_{\alpha}(A_T(x)) &= \inf_{t > 0} \left\{ (1 - \alpha)t + \int_t^T (z - t) \mu(z) dz + (T + b - t) \int_T^{\infty} \mu(z) dz \right\} \\ &= \inf_{t > 0} \left\{ (1 - \alpha)t + \int_t^T z \mu(z) dz + (T + b) q_T - t(1 - M(t)) \right\}. \end{aligned} \quad (5.10)$$

Differentiating (5.10) with respect to t gives

$$(1 - \alpha) - (1 - M(t)) = M(t) - \alpha.$$

Since the second derivative of (5.10) equals $\mu(t) \geq 0$, the minimizer is any t^* satisfying $M(t^*) = \alpha$. If M is continuous and strictly increasing, then $t^* = M^{-1}(\alpha)$ is unique. Substituting t^* and using $1 - M(t^*) = 1 - \alpha$ yields

$$\text{CVaR}_{\alpha}(A_T(x)) = \frac{1}{1 - \alpha} \left(\int_{t^*}^T z \mu(z) dz + (T + b) q_T \right). \quad (5.11)$$

Case 2: $T \leq t \leq T + b$. Here $(A_T(x) - t)^+ = 0$ for $x < T$ and $(T + b - t)$ for $x \geq T$. Since (5.3) is linear in t , the minimum occurs at one of the endpoints of the range of t . As a result, we have

$$\text{CVaR}_\alpha(A_T(x)) = \min \left\{ T + b \frac{q_T}{1 - \alpha}, T + b \right\}. \quad (5.12)$$

Case 3: $t \geq T + b$. In this range $(A_T(x) - t)^+ = 0$ for all x , so the infimum is attained at $t = T + b$, yielding

$$\text{CVaR}_\alpha(A_T(x)) = T + b.$$

Combining the above cases, $\text{CVaR}_{\alpha, \mu}[A_T(x)]$ is the minimum between (5.11) (Case 1) and (5.12) (Case 2), which completes the proof.

Analysis of extreme cases.

(i) For $\alpha = 0$, we have $t^* = \inf \text{supp}(\mu) = \ell$. From (5.11),

$$\text{CVaR}_0(A_T(x)) = \int_{\ell}^T z \mu(z) dz + (T + b)q_T = \mathbb{E}[A_T(x)],$$

since $A_T(x) = x$ on $\{x < T\}$ and $A_T(x) = T + b$ on $\{x \geq T\}$. Moreover, $\text{CVaR}_0(A_T)$ is equal to (5.11), because

$$\int_{\ell}^T z \mu(z) dz + (T + b)q_T \leq T(1 - q_T) + (T + b)q_T = T + bq_T.$$

(ii) Suppose that $\alpha \rightarrow 1$, then we distinguish between cases $q_T > 0$ and $q_T = 0$. If $q_T > 0$, both (5.11) and the first term of (5.12) contain $(1 - \alpha)^{-1}$ times a positive quantity; the minimum is therefore the remaining term $T + b$, so

$$\lim_{\alpha \rightarrow 1} \text{CVaR}_\alpha(A_T(x)) = T + b.$$

If $q_T = 0$ (equivalently $T > u := \sup \text{supp}(\mu)$), then (5.11) reduces to

$$\frac{1}{1 - \alpha} \int_{t^*}^T z \mu(z) dz = \frac{1}{1 - \alpha} \int_{t^*}^u z \mu(z) dz = \mathbb{E}[x \mid x \geq t^*],$$

since $\mathbb{P}[x \geq t^*] = 1 - \alpha$ and $\mu \equiv 0$ on $(u, T]$. As $\alpha \rightarrow 1$, $t^* \rightarrow u$ and bounded convergence yields $\mathbb{E}[x \mid x \geq t^*] \rightarrow u$. In the same regime, (5.12) becomes

$\min\{T, T + b\} = T \geq u$, so the minimum is given by (5.11) and

$$\lim_{\alpha \rightarrow 1} \text{CVaR}_\alpha(A_T(x)) = u.$$

Combining the above we obtain that $\lim_{\alpha \rightarrow 1} \text{CVaR}_\alpha(A_T(x)) = \max\{u, T + b\}$. ■

Theorem 5.4 captures the tradeoff between optimizing for average-case performance, on the one hand, and safeguarding against adversarial prediction, on the other hand. Specifically, if $\alpha = 0$, the objective reduces to minimizing the expected cost, since

$$\int_0^T z \cdot \mu(z) dz + (b + T) \cdot q_T = \mathbb{E}_{z \sim \mu}[A_T(z)].$$

In contrast, if $\alpha \rightarrow 1$, two cases arise: if $q_T > 0$, then from (5.9), $\text{CVaR}_{\alpha, \mu}[A_T(x)] = b + T$, whereas if $q_T = 0$, then $\text{CVaR}_{\alpha, \mu}[A_T(x)] = u := \sup \text{supp}(\mu)$. Hence, in the fully risk-averse limit,

$$\text{CVaR}_{\alpha, \mu}[A_T(x)] = \min\{b + T, u\}.$$

5.4 Application in 1-max search

We recall the definitions of the 1-Max Search problem, which was introduced in Section 2.1.3. In this problem, the input is a sequence σ of *prices* in $[1, M]$, where M is known to the algorithm. We denote by x_σ the maximum price in σ , or simply by x when σ is implied. Any online algorithm is a *threshold* algorithm, in that it selects some $T \in [1, M]$ and accepts the first price in σ that is at least T . If such a price does not exist in σ , then the profit of the algorithm is defined to be 1. We denote by A_T an online algorithm A with threshold T , and by $A_T(\sigma)$ its profit on input σ . In a learning-augmented setting, the online algorithm has access to a *prediction* y , and the prediction error is defined as $\eta = |x_\sigma - y|$. We write the prediction range as $R_y = [\ell, u]$ (with $\ell, u \in [1, M]$); when an error bound is assumed, we use $R_y = [(1 - \delta)y, (1 + \delta)y]$ with $\delta \in [0, 1]$ (the δ -tolerant setting).

For any $r \geq \sqrt{M}$, algorithm A_T is r -robust if and only if $T \in [t_1, t_2]$, where $t_1 = M/r$ and $t_2 = r$ [Sun+21]. Here, recall that $\sqrt{M}$ is the optimal competitive ratio without predictions [El-98]. Note that there is an infinite number of r -robust algorithms, hence our objective is:

Objective. For each of the decision-theoretic models of Section 5.2, and a given robustness requirement r , find the optimal threshold T that optimizes the cor-

responding measure. We denote by $T_{\max}^*$, T_{avg}^* and T_{CVaR}^* the optimal threshold values.

5.4.1 Distance measures

We first describe the ideal solution. Its performance follows from the definition of t_1 and t_2 . See Figure 5.2 for an illustration.

► **Lemma 5.5.** Given a robustness requirement r , and a sequence σ , the ideal algorithm I_r chooses the threshold $\min\{t_2, \max\{t_1, x_\sigma\}\}$. Its performance ratio is

$$\text{perf}(I_r, \sigma) = \begin{cases} x_\sigma, & \text{if } x_\sigma \in [1, t_1), \\ 1, & \text{if } x_\sigma \in [t_1, t_2], \\ \frac{x_\sigma}{t_2}, & \text{if } x_\sigma \in (t_2, M]. \end{cases} \quad (5.13)$$

◀

Proof. To be r -robust, any algorithm must choose a threshold $T \in [t_1, t_2]$, where $t_1 = M/r$ and $t_2 = r$.

Case 1: $x_\sigma < t_1$. Then I_r sets $T = t_1$ and never accepts, so $A_{t_1}(\sigma) = 1$. The offline optimum is x_σ , hence $\text{perf}(I_r, \sigma) = x_\sigma$. Any other r -robust A_T has $T \geq t_1$ and still returns 1, thus also attains ratio x_σ . Therefore I_r is optimal here.

Case 2: $x_\sigma \in [t_1, t_2]$. Set $T = x_\sigma$, hence $A_T(\sigma) = x_\sigma$ and $\text{perf}(I_r, \sigma) = 1$. Any r -robust A_T with $T > x_\sigma$ returns 1 (ratio $x_\sigma > 1$); with $T < x_\sigma$, if a value $v \in [T, x_\sigma)$ appears before x_σ , then the ratio is $x_\sigma/v > 1$; otherwise it accepts x_σ and ties I_r .

Case 3: $x_\sigma > t_2$. Set $T = t_2$, so $A_{t_2}(\sigma) \geq t_2$ and $\text{perf}(I_r, \sigma) \leq x_\sigma/t_2$. Any r -robust algorithm has $T \leq t_2$. If $T < t_2$, consider $\sigma = (v, t_2)$ with $v \in [T, t_2)$; then $A_T(\sigma) = v$ and the ratio is $x_\sigma/v \geq x_\sigma/t_2$. Thus no r -robust algorithm achieves a smaller ratio than I_r .

Combining the three cases yields the stated threshold and performance. ■

From (5.1) and Lemma 5.5, it follows that for any $T \in [t_1, t_2]$,

$$d_{\max}(A_T) = \sup_{\sigma} \sup_{x \in R_y} \left(\frac{x}{A_T(\sigma)} - \text{perf}(I_r, \sigma) \right) \cdot w(x). \quad (5.14)$$

We first give an analytical solution in the *unweighted setting*, i.e., when w is the unit function.

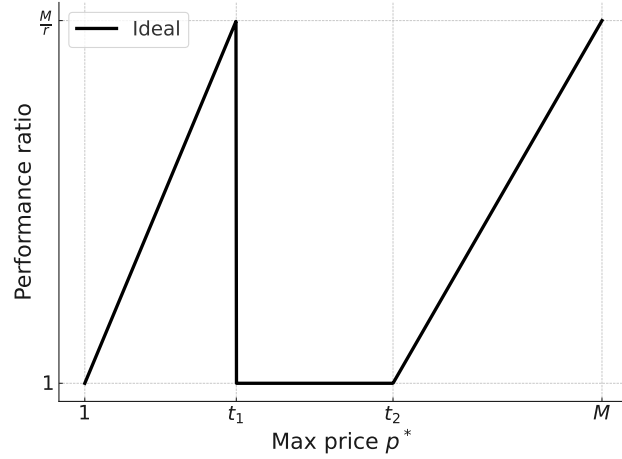


Figure 5.2: Ideal performance of 1-Max Search with robustness requirement r .

► **Theorem 5.6.** For uniform weights ($w = 1$), for all $x \in R_y = [\ell, u]$,

$$T_{\max}^* = \begin{cases} \min\{t_2, \max\{t_1, \sqrt{u}\}\}, & \text{if } u \leq t_2, \\ \min\{t_2, \max\{t_1, \tilde{T}\}\}, & \text{otherwise,} \end{cases}$$

where $\tilde{T} = t_2 - u + \sqrt{(u - t_2)^2 + 4t_2^2 u}$. ◀

Proof. The prediction range $R_y = [\ell, u]$ may not overlap with $[t_1, t_2]$. If $u \leq t_1$, then $T_{\max}^* = t_1$ and $d_{\max}(A_T) = 0$. If $\ell \geq t_2$, then $T_{\max}^* = t_2$ and again $d_{\max}(A_T) = 0$.

Now assume R_y intersects $[t_1, t_2]$.

Case 1: $\ell, u \in [t_1, t_2]$. Then $\text{perf}(I_r, x) = 1$ for all $x \in R_y$. (i) If $T \leq \ell$, then $d_{\max}(A_T) = u/T - 1$ (adversary uses $x = u$). (ii) If $T \in [\ell, u]$, then $d_{\max}(A_T) = \max\{u/T - 1, T - 1\}$. (iii) If $T \geq u$, then $d_{\max}(A_T) = u - 1$. Minimizing the worst of the two terms in (ii) gives $T = \sqrt{u}$. Enforcing $T \in [t_1, t_2]$ yields $T_{\max}^* = \max\{t_1, \min\{t_2, \sqrt{u}\}\}$.

Case 2: $t_1 \leq \ell$ and $u \geq t_2$. Here $\text{perf}(I_r, x)$ may exceed 1. It suffices to choose $T \in [\ell, t_2]$, where

$$d_{\max}(A_T) = \max \left\{ \frac{u}{T} - \frac{u}{t_2}, T - 1, \frac{t_2}{T} - 1 \right\}.$$

Balancing $\frac{u}{T} - \frac{u}{t_2} = T - 1$ gives

$$T = t_2 - u + \sqrt{(u - t_2)^2 + 4t_2^2 u}.$$

Projecting onto $[t_1, t_2]$ yields $T_{\max}^* = \min\{t_2, \max\{t_1, T\}\}$. ■

For some intuition behind Theorem 5.6, we note that in the first case, the algorithm aims to minimize the distance from the line $y(x) = 1$ (the ideal performance). In this case, the threshold has a dependency on $\sqrt{u}$, as derived from an analysis similar to the competitive ratio (which is equal to the square root of the maximum price). In the second case, the algorithm aims to minimize the distance from a more complex ideal performance, which includes two line segments. This explains the dependency on the more complex value $\tilde{T}$. One can also show that $\tilde{T} \geq \sqrt{u}$: this is explained intuitively, since in the second case, the algorithm has more “leeway”, given that the ideal performance ratio attains higher values.

The case of general weight functions is more complex computationally. We can formulate it as a two-person zero-sum game between the algorithm (choosing T) and the adversary (choosing x). If $R_y \subseteq [t_1, t_2]$, the payoff is

$$\max \left\{ \max_{x \geq T} \left(\frac{x}{T} - 1 \right) w(x), \max_{x < T} (T - 1) w(x) \right\},$$

since here $\text{perf}(I_r, x) = 1$. Otherwise,

$$\begin{aligned} \max \left\{ \max_{T \leq x \leq t_2} \left(\frac{x}{T} - 1 \right) w(x), \right. \\ \max_{x \geq t_2} \left(\frac{x}{T} - \frac{x}{t_2} \right) w(x), \\ \left. \max_{x < T} (T - 1) w(x) \right\}, \end{aligned} \quad (5.15)$$

by (5.13). In general, a closed form is not available for arbitrary w , but certain simple families (e.g., linear weights) admit analytic solutions.

We next show how to compute $T_{\max}^*$ for a linear weight function, defined as:

$$w(x) = \max \left\{ 0, 1 - \frac{|x - y|}{\delta} \right\}. \quad (5.16)$$

► **Example 5.7.** We will compute $T_{\max}^*$ for a linear weight function,

$$w(x) = \max \left\{ 0, 1 - \frac{|x - y|}{y\delta} \right\}. \quad (5.17)$$

For simplicity, we only show the computation for the case $R_y \subseteq [t_1, t_2]$. The other cases can be handled along similar lines, using (5.15). In this case, the payoff is

$$\max \begin{cases} (x-1) \left(1 - \frac{y-x}{y\delta}\right), & \text{if } x < T \text{ and } x \in [(1-\delta)y, y], \\ \left(\frac{x}{T} - 1\right) \left(1 - \frac{y-x}{y\delta}\right), & \text{if } x \geq T \text{ and } x \in [(1-\delta)y, y], \\ (x-1) \left(1 - \frac{x-y}{y\delta}\right), & \text{if } x < T \text{ and } x \in [y, (1+\delta)y], \\ \left(\frac{x}{T} - 1\right) \left(1 - \frac{x-y}{y\delta}\right), & \text{if } x \geq T \text{ and } x \in [y, (1+\delta)y]. \end{cases}$$

We denote these four expressions by e_1, e_2, e_3, e_4 , respectively. We analyze the adversary's best response for a fixed threshold T .

Case A: $T \leq y$.

Subcase A1: $(1-\delta)y \leq x \leq T$. e_1 is concave in x (second derivative $2/(y\delta)$), so the maximum is at an endpoint. Since $e_1((1-\delta)y) = 0$ and $e_1(T) = (T-1)(T - (1-\delta)y)/(y\delta) > 0$, the best response is $x = T$, giving

$$v_1 = (T-1) \frac{T - (1-\delta)y}{y\delta}.$$

Subcase A2: $T \leq x \leq y$. e_2 is concave (second derivative $2/(y\delta T)$). At the endpoints, $e_2(T) = 0$ and $e_2(y) = y/T - 1 \geq 0$, so the best response is $x = y$, yielding

$$v_2 = \frac{y}{T} - 1.$$

Subcase A3: $y \leq x \leq (1+\delta)y$. e_4 is concave (second derivative $-2/(y\delta T)$) and maximized at the stationary point $x = (T + (1+\delta)y)/2 \in [y, (1+\delta)y]$, giving

$$v_4 = \frac{((1+\delta)y - T)^2}{4y\delta T}.$$

Summary of Case A. $v_2 \leq v_4$, hence the value is $\max\{v_1, v_4\}$.

Case B: $T \geq y$.

Subcase B1: $(1 - \delta)y \leq x \leq y$. As in A1, e_1 is maximized at $x = y$, giving

$$e_1(y) = y - 1.$$

Subcase B2: $y \leq x \leq T$. e_3 is concave in x and is maximized at $x = y$, yielding

$$v_3 = y - 1.$$

(Note v_3 matches the value in B1 and is independent of T .)

Subcase B3: $T \leq x \leq (1 + \delta)y$. As in A3, the value is v_4 .

Summary of Case B. For $T \in [y, (1 + \delta)y]$, the value is $\max\{v_3, v_4\}$. Using $1 \leq (1 - \delta)y$,

$$v_3 - v_4((1 - \delta)y) = y - 1 - \frac{y\delta}{(1 - \delta)y} \geq y - 1 - y\delta \geq 0,$$

so $v_3 \geq v_4$ over $T \in [(1 - \delta)y, (1 + \delta)y]$.

Summary of both cases. If $T \geq y$, the value is $v_3 = y - 1$. For $T \leq y$, we already have $v_4 \leq v_3$, and

$$v_1 = (T - 1) \frac{T - (1 - \delta)y}{y\delta} \leq (y - 1) \cdot 1 = v_3,$$

so $\max\{v_1, v_4\} \leq v_3$. Thus the algorithm's best strategy is to pick $T \in [(1 - \delta)y, y]$ such that $v_1(T) = v_4(T)$. The closed form is unwieldy, so we omit it; see Figure 5.3. 

Next, we optimize the average distance. Using (5.2) and (5.13),

$$d_{\text{avg}}(A_T) = \begin{cases} \frac{1}{2y\delta} \left(\int_{(1-\delta)y}^T (x-1) \cdot w(x) dx + \int_T^{(1+\delta)y} \left(\frac{x}{T} - 1\right) \cdot w(x) dx \right), & \text{if } (1 + \delta)y \leq t_2, \\ \frac{1}{2y\delta} \left(\int_{t_1}^T (x-1) \cdot w(x) dx + \int_T^{t_2} \left(\frac{x}{T} - 1\right) \cdot w(x) dx + \int_{t_2}^{(1+\delta)y} \left(\frac{x}{T} - \frac{x}{t_2}\right) \cdot w(x) dx \right), & \text{otherwise.} \end{cases} \quad (5.18)$$

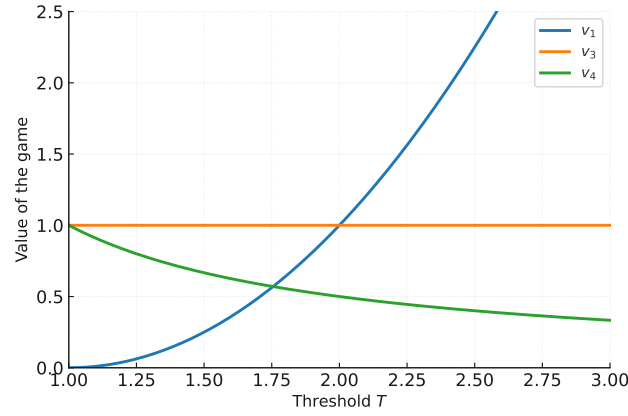


Figure 5.3: Illustration of the different values of the game in Example 5.7: v_1 (blue), v_3 (orange), and v_4 (green). Parameters: $y = 2$, $\delta = 0.5$.

► **Example 5.8.** We illustrate the computation for the linear weight (5.17) in the first case of (5.18) (i.e., $(1 + \delta)y \leq t_2$). Since $w(x)$ increases on $[(1 - \delta)y, y]$ and decreases on $[y, (1 + \delta)y]$, we split by $T < y$ vs. $T \geq y$.

If $T < y$:

$$\begin{aligned}
 d_{\text{avg},1}(T) &= \frac{1}{2y\delta} \left(\int_{(1-\delta)y}^T (x-1) \left(1 - \frac{y-x}{y\delta}\right) dx + \int_T^y \left(\frac{x}{T} - 1\right) \left(1 - \frac{y-x}{y\delta}\right) dx \right. \\
 &\quad \left. + \int_y^{(1+\delta)y} \left(\frac{x}{T} - 1\right) \left(1 - \frac{x-y}{y\delta}\right) dx \right) \\
 &= \frac{1}{12(y\delta)^2 T} \left(- (y\delta)^3 (T-1) + 3(y\delta)^2 ((y-2)T + y) \right. \\
 &\quad \left. + 3y\delta(T-1)(T^2 - y^2) + (T-1)(y-T)^2(y+2T) \right).
 \end{aligned}$$

If $T \geq y$:

$$\begin{aligned}
 d_{\text{avg},2}(T) &= \frac{1}{2y\delta} \left(\int_{(1-\delta)y}^y (x-1) \left(1 - \frac{y-x}{y\delta}\right) dx \right. \\
 &\quad \left. + \int_y^T \left(\frac{x}{T} - 1\right) \left(1 - \frac{x-y}{y\delta}\right) dx \right)
 \end{aligned}$$

$$\begin{aligned}
& + \int_T^{(1+\delta)y} \left(\frac{x}{T} - 1 \right) \left(1 - \frac{x-y}{y\delta} \right) dx \\
& = \frac{y\delta + 3y - (6 + y\delta - 3y)T}{12T}.
\end{aligned}$$

The optimal threshold T_{avg}^* is obtained by minimizing $d_{\text{avg}}(T)$: solve each case via second-order analysis and take the better of the two values. ◀

5.4.2 Risk-Based Analysis

We consider the setting where the algorithm has access to a distributional prediction μ with support $R_y = [\ell, u] = [(1-\delta)y, (1+\delta)y]$, for some $\delta \in [0, 1]$. Given a robustness requirement r and a risk parameter $\alpha \in [0, 1)$, we seek an r -robust algorithm that minimizes the α -consistency. Since this is a profit-maximization problem, we use the profit version of CVaR and the resulting α -consistency:

$$\text{CVaR}_\alpha(X) = \sup_t \left\{ t - \frac{1}{1-\alpha} \mathbb{E}[(t-X)^+] \right\} \quad \text{and} \quad \alpha\text{-cons}(A) = \sup_{F \in \mathcal{F}} \frac{\mathbb{E}_{\sigma \sim F}[\text{OPT}(\sigma)]}{\text{CVaR}_{\alpha, F}(A(\sigma))}.$$

We first show that the α -consistency is determined by a worst-case input distribution F^* : draw $x \sim \mu$ and present a sequence σ whose values increase infinitesimally from 1 up to x , after which a final price 1 appears. For a threshold algorithm A_T , this ensures $A_T(\sigma) \in \{1, T\}$.

► **Lemma 5.9.** For any threshold algorithm A_T ,

$$\alpha\text{-cons}(A_T) = \frac{\mathbb{E}_{x \sim \mu}[x]}{\text{CVaR}_{\alpha, F^*}[A_T(\sigma)]}.$$
◀

Proof. From the definition of F^* , it follows that

$$\mathbb{E}_{p \sim \mu}[p] = \mathbb{E}_{p \sim F^*}[p],$$

hence the α -consistency of A_T is at least the RHS of the equation in Lemma 5.9.

Let $\tilde{F}$ be a distribution that maximizes the α -consistency, then it must be that

$$\alpha\text{-cons}(A_T) \geq \frac{\mathbb{E}_{p \sim \mu}[p]}{\text{CVaR}_{\alpha, \tilde{F}}[A_T(\sigma)]}.$$

We can argue that $\text{CVaR}_{\alpha, \tilde{F}}[A_T(\sigma)] \leq \text{CVaR}_{\alpha, F^*}[A_T(\sigma)]$. This follows directly from (5.3), and the observation that in any sequence σ in the support of F^* , we have that $A_T(\sigma) = 1$, if $x_\sigma < T$, and $A_T(\sigma) = T$, if $x_\sigma \geq T$. Hence, we also showed that the α -consistency is at least the RHS of the equation in Lemma 5.9, which concludes the proof. ■

Since the numerator in Lemma 5.9 is independent of the algorithm, it suffices to maximize $\text{CVaR}_{\alpha, F^*}[A_T(\sigma)]$ over r -robust thresholds $T \in [t_1, t_2]$, where $t_1 = M/r$ and $t_2 = r$. Let

$$q_T = \mathbb{P}_{\sigma \sim F^*}[A_T(\sigma) = 1] = \mathbb{P}_{x \sim \mu}[x < T] = \int_{\ell}^{\min\{T, u\}} \mu(z) dz,$$

so that $A_T(\sigma) = T$ with probability $1 - q_T$.

► **Theorem 5.10.** Let $\ell = (1 - \delta)y$. Then

$$T_{\text{CVaR}}^* = \arg \max_{T \in [t_1, t_2]} S(T) \quad \text{where} \quad S(T) = \max \left\{ \frac{T(1 - \alpha - q_T) + q_T}{1 - \alpha}, \ell \right\}.$$

◀

Proof. Similar to ski rental, the computation of $\text{CVaR}_{\alpha, \mu}[A_T(\sigma)]$ for the 1-max search problem requires a case analysis based on the parameter t of (5.3). Define $q_T = \mathbb{P}_{\sigma \sim F^*}[A_T(\sigma) = 1]$ as the probability that the algorithm A_T selects the value 1, and $1 - q_T$ as the probability it selects the threshold T . Recall that these are the only two possibilities, from the definition of F^* , without any assumptions of R_y . Under the assumption that $R_y = [(1 - \delta)y, (1 + \delta)y]$, we can obtain a better lower bound for A_T , i.e., we know it can ensure a minimum profit of $(1 - \delta)y$. We proceed with the analysis of this setting, and consider the following cases.

Case 1: $t \geq T$. Then

$$\text{CVaR}_{\alpha, \mu}[A_T(\sigma)] = \sup_{t \geq T} \left\{ -t \left(\frac{\alpha}{1 - \alpha} \right) + \frac{q_T(1 - T) + T}{1 - \alpha} \right\}.$$

In this case, the optimal value of t is equal to T , hence we obtain:

$$\text{CVaR}_{\alpha, \mu}[A_T(\sigma)] = \frac{T(1 - q_T - \alpha) + q_T}{1 - \alpha}.$$

Case 2: $t \leq (1 - \delta)y$. In this case, $(t - A_T(\sigma))^+ = 0$, and

$$\text{CVaR}_{\alpha,\mu}[A_T(\sigma)] = (1 - \delta)y.$$

Case 3: $t \in [(1 - \delta)y, T]$. Then

$$(t - A_T(\sigma))^+ = \begin{cases} 0, & \text{w. p. } 1 - q_T, \\ t - 1, & \text{w. p. } q_T, \end{cases}$$

from which we get that

$$\text{CVaR}_{\alpha,\mu}[A_T(\sigma)] = \sup_{t \in [(1-\delta)y, T]} \left\{ t \left(\frac{1 - \alpha - q_T}{1 - \alpha} \right) + \frac{q_T}{1 - \alpha} \right\}.$$

We consider two further subcases. If $1 - \alpha - q_T \leq 0$, then we obtain

$$\text{CVaR}_{\alpha,\mu}[A_T(\sigma)] = (1 - \delta)y.$$

In the case, when $1 - \alpha - q_T > 0$, we have that

$$\text{CVaR}_{\alpha,\mu}[A_T(\sigma)] = \frac{T(1 - q_T - \alpha) + q_T}{1 - \alpha}.$$

Combining all the above cases, it follows that:

$$\text{CVaR}_{\alpha,\mu}[A_T(\sigma)] = \max \left\{ \frac{T(1 - q_T - \alpha) + q_T}{1 - \alpha}, (1 - \delta)y \right\}.$$

■

Theorem 5.10 interpolates between two cases. When $\alpha = 0$, the algorithm maximizes its expected profit, assuming that the maximum price in the input sequence has distribution μ . In this case, we find an r -robust algorithm of optimal consistency, under the distributional setting of [Dia+21] which is a novel contribution for the 1-max search problem by itself. When $\alpha \rightarrow 1$, the online algorithm must choose its threshold under the assumption that the maximum price is chosen adversarially within the support $[\ell, u]$ of μ , hence the threshold (and the algorithm's profit) is $\ell = (1 - \delta)y$. This recovers the analysis of the δ -tolerant algorithm in [AKZ22].

5.5 Application in contract scheduling

In this section, we apply our framework to the contract scheduling problem³, which was introduced in Section 2.1.3. We briefly recall the setting. A schedule is defined as an increasing sequence $X = (x_i)_{i \in \mathbb{N}}$, where x_i denotes the *length* of the i -th *contract*. These lengths determine the durations allocated to sequential runs of a contract algorithm. The completion time of the i -th contract is given by $\sum_{j=0}^i x_j$. Given an interruption time T , let $\ell(X, T)$ denote the length of the longest contract completed by time T . The key performance measure, the *acceleration ratio* [RZ91], is defined as

$$\text{acc}(X) = \sup_T \text{perf}(X, T), \quad \text{where } \text{perf}(X, T) = \frac{T}{\ell(X, T)}. \quad (5.19)$$

It is known that the best-possible acceleration ratio is equal to 4, which is attained by any *doubling* schedule of the form $X_\lambda = (\lambda 2^i)_i$, where $\lambda \in [1, 2)$. In fact, under very mild assumptions, doubling schedules are essentially the unique schedules that optimize the acceleration ratio. However, according to the definition in (5.19), no schedule can guarantee a bounded ratio if the interruption time can be arbitrarily small. To address this, we assume the schedule is *bi-infinite*, meaning it includes an infinite number of infinitesimally small contracts at the beginning. For instance, the doubling schedule can be extended to $(2^i)_{i \in \mathbb{Z}}$, where the completion time of contract $i \geq 0$ is given by $\sum_{j=-\infty}^i 2^j = 2^{i+1}$. This ensures consistent behavior even for extremely small interruption times. See [Ang+24a] for further discussion of this modeling choice. Our objective in this section can thus be summarized as follows:

Objective: For each of the decision-theoretic models of Section 5.2, find $\lambda \in [1, 2)$ such that the schedule X_λ optimizes the corresponding measure.

We will denote by $\lambda_{\max}^*$, λ_{avg}^* and λ_{cvar}^* the optimal values according to the maximum/average distance, and according to CVaR, respectively. Given a schedule X_λ , we will use the notation $k_\lambda(t)$ to denote the index of the largest contract in X_λ that completes by time t , hence $k_\lambda(t) = \lfloor \log_2 \frac{t}{\lambda} \rfloor$.

5.5.1 Distance measures

Here, we consider the setting in which there is a prediction y on the interruption time. We begin with identifying an ideal schedule, which, in the context of contract

³ This is a problem of incomplete information, which can be considered as an online problem, in the sense that in each time step the scheduler must decide whether to continue the current contract, or start a new one.

schedule, is a 4-robust schedule X that optimizes the length $\ell(X, y)$, i.e., the length of the contract completed by the predicted time y . From [AK23a], we know that such an ideal schedule completes a contract of length $y/2$, precisely at time y ; furthermore no 4-robust schedule can complete a longer contract by y . We thus have the following.

► **Remark 5.11.** Let I_4 denote the ideal 4-robust contract schedule. Then, $\text{perf}(I_4, T) = T/2$, for all T . ◀

From (5.1), (5.19) and Remark 5.11 it follows that the maximum distance of a schedule X_λ can be expressed as

$$d_{\max}(X_\lambda) = \sup_{T \in R_y} \left(\frac{T}{\ell(X_\lambda, T)} - 2 \right) \cdot w(T), \quad (5.20)$$

where recall that R_y is the range of the prediction y .

Algorithm 7 shows how to compute $\lambda_{\max}^*$. We give the intuition behind the algorithm. We prove, in Theorem 5.12, that the distance can be maximized only at a discrete set of times, denoted by S . This set includes the prediction y , the last time a contract in X_λ completes prior to y , and an additional set of times, denoted by S' which are the roots of a differential equation, defined in step 2 of the algorithm. To show this, we rely on two facts: that w is piece-wise monotone (i.e., bitonic), and that the performance function of any 4-robust schedule X_λ is piece-wise linear, with values in $[2, 4]$.

► **Theorem 5.12.** Algorithm 7 returns an optimal schedule according to $d_{\max}$. ◀

Proof. Recall that the performance ratio of the schedule $X_\lambda = (\lambda 2^i)_i$ is expressed as

$$\text{perf}(X_\lambda, T) = \frac{T}{\ell(X_\lambda, T)} = \frac{T}{\lambda \cdot 2^{\lfloor \log_2 \frac{T}{\lambda} \rfloor - 1}}.$$

We observe that $\text{perf}(X_\lambda, T)$ is a piece-wise linear function. Specifically, if T belongs in the interval $(\lambda 2^j, \lambda 2^{j+1}]$, then $\text{perf}(X_\lambda, T)$ is a linear increasing function, with value equal to 2, at $T = \lambda 2^j + \varepsilon$, and value equal to 4 at $T = \lambda 2^{j+1}$, where ε is an infinitesimally small, positive value. This linear growth arises from the structure of the schedule, which starts a new contract at the endpoint of each interval, as illustrated in Figure 5.4. For this reason, $\text{perf}(X_\lambda, T)$ has a discontinuity at the endpoint of each interval.

By definition, y belongs to the interval $(T_{k_\lambda(y)}, T_{k_\lambda(y)+1}]$. To simplify the notation, in the remainder of the proof we use k to denote $k_\lambda(y)$. We claim $d_{\max}(X_\lambda)$ is

Algorithm 7: Algorithm for computing $\lambda_{\max}^*$ **Input:** Prediction y with range R_y , weight function w **Output:** The optimal value $\lambda_{\max}^*$ 1 Define the set of *critical* times as

$$S = \{y, \lambda \cdot 2^{k_\lambda(y)}\} \cup S',$$

where S' is the set of all solutions to the differential equation

$$w(T) + w'(T) \cdot \left(T - \lambda \cdot 2^{\lfloor \log_2 \frac{T}{\lambda} \rfloor}\right) = 0$$

2 **foreach** $T \in S$ **do**3 Compute $d(X_\lambda, T) = \left(\frac{T}{\lambda \cdot 2^{k_\lambda(T)-1}} - 2\right) \cdot w(T)$;4 **Return** $\lambda_{\max}^* = \arg \min_{\lambda \in [1,2)} \max_{T \in S} d(X_\lambda, T)$;

maximized for some $T \in [T_k, T_{k+1}]$, specifically at one of a finite set of critical points S . To establish this claim, we make the following observations:

- At $T = T_k$, the performance ratio reaches its maximum value equal to 4, for the entire interval $(T_{k-1}, T_k]$.
- Any $T > T_{k+1}$ or $T < T_k$ does not need to be considered in the computation of $d_{\max}$, due to the monotonicity of the weight function, and the structural properties of the schedule X_λ , as discussed above.

Given that the bitonic nature of the weight function, we observe that for all $T < y$, $w(T)$ is non-decreasing, hence within the interval $[T_k, y)$, it suffices to only consider T_k as a maximizing candidate. Furthermore, in the interval $[y, T_{k+1}]$, $w(T)$ is non-increasing, while the performance ratio grows linearly. Thus, one must find the local maxima for $T \in (y, T_{k+1}]$, by solving $d'(X_\lambda, T) = 0$, or equivalently

$$w(T) + w'(T) \cdot (T - \lambda \cdot 2^{\lfloor \log_2 \frac{T}{\lambda} \rfloor}) = 0.$$

We thus show that it suffices to consider the set S as potential maximizers of the distance, as defined in Algorithm 7. ■

► **Corollary 5.13.** For the unit weight function $w(t) = 1$, and $R_y = [(1 - \delta)y, (1 + \delta)y]$, the schedule that minimizes $d_{\max}$ is the schedule of [AK23a]. ◀

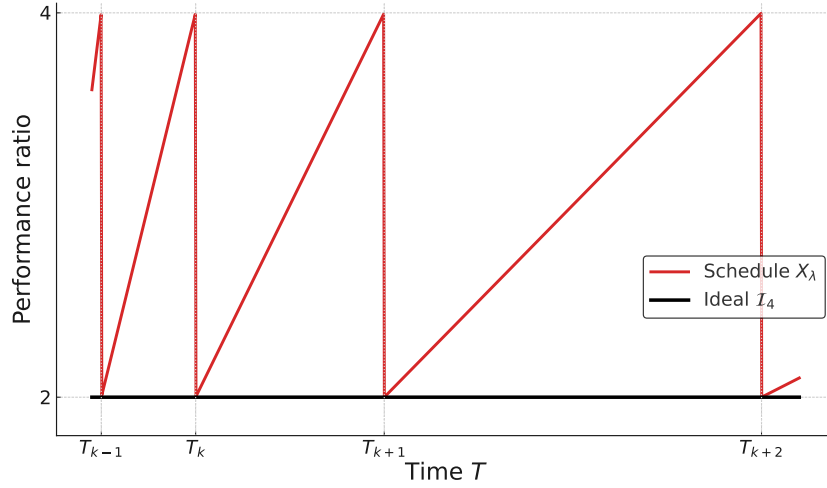


Figure 5.4: Performance ratio of the schedule X_λ compared to the ideal schedule I_4 .

Proof. The proof is a special case of the proof of Theorem 5.12. In this case, $d'(X_\lambda, T) = 1 > 0$, which implies that only local maxima for $d(X_\lambda, T)$ can occur at $T = T_{k+1}$ or at $(1 + \delta)y$.

We will consider two cases. First, suppose that $\delta > 1/3$. In this case, any schedule X_λ is such that $d_{\max}(X_\lambda) = 2$. This is because X_λ completes at least one contract within the time interval $[(1 - \delta)y, (1 + \delta)y]$.

For the second case, suppose that $\delta \leq 1/3$. Then, in order to minimize $d_{\max}$, and without loss of generality, λ must be chosen so that no contract terminates anywhere in $[(1 - \delta)y, (1 + \delta)y]$, since otherwise X_λ would have a performance ratio as large as 4, hence distance as large as 2. With this into account, λ must be further chosen so that X_λ completes a contract at time $(1 - \delta)y$. This is because, in this case, $\text{perf}(X_\lambda, T)$ is increasing in T , for $T \in [(1 - \delta)y, (1 + \delta)y]$. Hence the optimal algorithm is precisely the δ -tolerant algorithm. ■

Next, we show how to optimize the average distance, which from (5.2), and (5.19) is equal to

$$d_{\text{avg}}(X) = \frac{1}{2h} \int_{T \in R_y} \left(\frac{T}{\lambda \cdot 2^{\lfloor \log_2 \frac{T}{\lambda} \rfloor - 1}} - 2 \right) \cdot w(T) dT. \quad (5.21)$$

Optimizing (5.21) requires numerical methods. To ensure computational tractability, we impose a constraint on the range R_y of the prediction y . Specifically, we assume $\delta \leq \frac{1}{3}$. This assumption guarantees that for any schedule of the form $X = \lambda(2^i)_i$ there is at most one completed contract within R_y .

The length of the largest completed contract in X before $(1 - \delta)y$ is then given by $\lambda 2^{k_\lambda((1-\delta)y)-1}$. Using this, we divide the range R_y into two sub-intervals:

1. $[(1 - \delta)y, \lambda 2^{k_\lambda((1-\delta)y)+1}]$: In this interval, the performance ratio is

$$\frac{T}{\ell(X, T)} = \frac{T}{\lambda 2^{k_\lambda((1-\delta)y)-1}}.$$

2. $[\lambda 2^{k_\lambda((1-\delta)y)+1}, (1 + \delta)y]$: In this interval, the performance ratio is

$$\frac{T}{\ell(X, T)} = \frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}}.$$

The average distance $d_{\text{avg}}(X)$ is then expressed as:

$$d_{\text{avg}}(X_\lambda) = \frac{1}{2y\delta} \left(\int_{(1-\delta)y}^{\lambda 2^{k_\lambda((1-\delta)y)+1}} \left(\frac{T}{\lambda 2^{k_\lambda((1-\delta)y)-1}} - 2 \right) \cdot w(T) dT \right. \quad (5.22)$$

$$\left. + \int_{\lambda 2^{k_\lambda((1-\delta)y)+1}}^{(1+\delta)y} \left(\frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}} - 2 \right) \cdot w(T) dT \right). \quad (5.23)$$

► **Example 5.14.** As an example, consider the case in which w is a bitonic linear function defined by

$$w(T) = \max \left\{ 0, 1 - \frac{|T - y|}{y\delta} \right\},$$

To apply this weight function in the computation of (5.23), we divide the prediction interval $R_y = [(1 - \delta)y, (1 + \delta)y]$ into three subintervals based on the structure of the schedule and function w :

- $T \in [(1 - \delta)y, \lambda 2^{k_\lambda((1-\delta)y)+1}]$: In this case, $\text{perf}(X, T) = \frac{T}{\lambda 2^{k_\lambda((1-\delta)y)-1}}$. Then,

$$\int_{(1-\delta)y}^{\lambda 2^{k_\lambda((1-\delta)y)+1}} \left(\frac{T}{\lambda 2^{k_\lambda((1-\delta)y)-1}} - 2 \right) \cdot \left(1 - \frac{y - T}{y\delta} \right) dT.$$

- $T \in [\lambda 2^{k_\lambda((1-\delta)y)+1}, y]$: In this case, $\text{perf}(X, T) = \frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}}$. Then,

$$\int_{\lambda 2^{k_\lambda((1-\delta)y)+1}}^y \left(\frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}} - 2 \right) \cdot \left(1 - \frac{y - T}{y\delta} \right) dT.$$

- $T \in [y, (1 + \delta)y]$: In this case, $\text{perf}(X, T) = \frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}}$. Then,

$$\int_y^{(1+\delta)y} \left(\frac{T}{\lambda 2^{k_\lambda((1-\delta)y)}} - 2 \right) \cdot \left(1 - \frac{y-T}{y\delta} \right) dT.$$

To summarize, we obtain from the above cases, and (5.23) that

$$d_{\text{avg}}(X) = \frac{-3(y\delta)^2\lambda + 4^{k_\lambda((1-\delta)y)+1}\lambda^3 + 3 \cdot 2^{k_\lambda((1-\delta)y)}\lambda^2 y(\delta-1)}{3(y\delta)^2\lambda} \\ + \frac{2^{-2-k_\lambda((1-\delta)y)} \left(-(y\delta)^3 + 9(y\delta)^2y - 3(y\delta)y^2 + y^3 \right)}{3(y\delta)^2\lambda}.$$

Optimizing in terms of λ via second-order analysis and solving for the derivative's root gives three solutions, but only one real root. Thus the optimized value is:

$$\lambda_{\text{avg}}^* = 2^{-3(1+k_\lambda((1-\delta)y))} \left(4^{k_\lambda((1-\delta)y)} y(1-\delta) \right. \\ \left. + \frac{16^{k_\lambda((1-\delta)y)} y^2(\delta-1)^2}{\left(-3 \cdot 64^{k_\lambda((1-\delta)y)} A + 4\sqrt{4096^{k_\lambda((1-\delta)y)} B_1 B_2} \right)^{1/3}} \right. \\ \left. + \left(-3 \cdot 64^{k_\lambda((1-\delta)y)} A + 4\sqrt{4096^{k_\lambda((1-\delta)y)} B_1 B_2} \right)^{1/3} \right).$$

where

$$A = 3(y\delta)^3 - 25(y\delta)^2y + 9(y\delta)y^2 - 3y^3, \\ B_1 = 5(y\delta)^3 - 39(y\delta)^2y + 15(y\delta)y^2 - 5y^3, \\ B_2 = (y\delta)^3 - 9(y\delta)^2y + 3(y\delta)y^2 - y^3.$$



5.5.2 Risk-based analysis

We now turn our attention to the CVaR analysis. Following the discussion of Section 5.2.2, the oracle provides the schedule with an imperfect distributional prediction μ . From (5.4), and the fact that any distributional prediction concerns only the interruption time (the only unknown in the problem), the α -consistency

of a schedule X_λ is equal to

$$\alpha\text{-cons}(A) = \frac{\mathbb{E}_{T \sim \mu}[T]}{\text{CVaR}_{\alpha, \mu}[\ell(X_\lambda, T)]}.$$

We thus seek X_λ that maximizes the Conditional Value-at-Risk of its largest completed contract for an interruption time drawn according to μ . To obtain a tractable expression of this quantity, we assume that μ has support $R_y = [(1 - \delta)y, (1 + \delta)y]$ with $\delta \leq \frac{1}{3}$. This captures the requirement that the support remains bounded, otherwise the distributional prediction becomes highly inaccurate. This implies that if t is drawn from μ , then in X_λ , $\ell(X_\lambda, t)$ can only have one of two possible values, namely $\lambda 2^{k_\lambda((1-\delta)y)-1}$ and $\lambda 2^{k_\lambda((1-\delta)y)}$.

Define $q_\lambda = \mathbb{P}[\ell(X_\lambda, T) = \lambda 2^{k_\lambda((1-\delta)y)-1}] = \int_{(1-\delta)y}^{\lambda 2^{k_\lambda((1-\delta)y)+1}} \mu(T) dT$, then from the discussion above we have that $\mathbb{P}[\ell(X_\lambda, T) = \lambda 2^{k_\lambda((1-\delta)y)}] = 1 - q_\lambda$. With this definition in place, we can find the optimal schedule.

► **Theorem 5.15.** Assuming $\delta \leq 1/3$, we have that

$$\text{CVaR}_{\alpha, \mu}[\ell(X_\lambda, T)] = \max \left\{ \frac{\lambda 2^{k_\lambda((1-\delta)y)-1}}{1 - \alpha} (2(1 - \alpha) - q_\lambda), \lambda 2^{k_\lambda((1-\delta)y)-1} \right\},$$

where $k_\lambda(t) = \lfloor \log_2 \frac{t}{\lambda} \rfloor$. Hence,

$$\lambda_{\text{CVaR}}^* = \arg \max_{\lambda \in [1, 2)} \text{CVaR}_{\alpha, \mu}[\ell(X_\lambda, T)],$$

◀

Theorem 5.15 interpolates between two extreme cases. If $\alpha = 0$, then our schedule maximizes the expected contract length assuming $T \sim \mu$, i.e., $\lambda 2^{k_\lambda((1-\delta)y)-1} \cdot q_\lambda + \lambda 2^{k_\lambda((1-\delta)y)} \cdot (1 - q_\lambda) = \lambda 2^{k_\lambda((1-\delta)y)-1} \cdot (2 - q_\lambda)$. This schedule recovers the optimal consistency in the standard case of a distributional prediction, as studied in [Ang+24a], and corresponds to a risk-seeking scheduler. In the other extreme, i.e., when $\alpha \rightarrow 1$, the schedule optimizes the length of a contract that completes by the time $(1 - \delta)y$, namely $\lambda 2^{k_\lambda((1-\delta)y)-1}$. We thus recover the consistency of the δ -tolerant schedule.

Proof. Recall the definition of the conditional value-at risk comes, as given in (5.3). In order to compute $\text{CVaR}_{\alpha, \mu}[\ell(X_\lambda, T)]$ we have to apply case analysis, based on the value of the parameter t :

Case 1: $t \geq \lambda 2^{k_\lambda((1-\delta)y)}$. Then

$$\text{CVaR}_{\alpha, \mu}[\ell(X_\lambda, T)] =$$

$$\sup_{t \geq \lambda 2^{k_\lambda((1-\delta)y)-1}} \left\{ t - \frac{1}{1-\alpha} \left(t - \lambda 2^{k_\lambda((1-\delta)y)-1} (2 - q_\lambda) \right) \right\}.$$

In this case, the optimal value of t is equal to $\lambda 2^{k_\lambda((1-\delta)y)-1}$, hence we obtain:

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \frac{\lambda 2^{k_\lambda((1-\delta)y)-1}}{1-\alpha} (2(1-\alpha) - q_\lambda).$$

Case 2: $t \leq \lambda 2^{k_\lambda((1-\delta)y)-1}$. In this case, $(t - \ell(X, T))^+ = 0$, and

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \lambda 2^{k_\lambda((1-\delta)y)-1}.$$

Case 3: $t \in [\lambda 2^{k_\lambda((1-\delta)y)-1}, \lambda 2^{k_\lambda((1-\delta)y)}]$. Then

$$(t - \ell(X, T))^+ = \begin{cases} 0, & \text{w. p. } 1 - q_\lambda, \\ t - \lambda 2^{k_\lambda((1-\delta)y)-1}, & \text{w. p. } q_\lambda, \end{cases}$$

from which we obtain that

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \sup_{t \in [\lambda 2^{k_\lambda((1-\delta)y)-1}, \lambda 2^{k_\lambda((1-\delta)y)}]} \left\{ t \left(1 - \frac{q_\lambda}{1-\alpha} \right) + \frac{\lambda 2^{k_\lambda((1-\delta)y)-1} q_\lambda}{1-\alpha} \right\}.$$

We consider two further subcases, based on whether $1 - \alpha - q_\lambda$ is positive or not. In the former case, we have that

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \frac{\lambda 2^{k_\lambda((1-\delta)y)-1}}{1-\alpha} (2(1-\alpha) - q_\lambda).$$

In the latter case, we obtain

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \lambda 2^{k_\lambda((1-\delta)y)-1}.$$

From the above case analysis, it follows that

$$\text{CVaR}_{\alpha,\mu}[\ell(X_\lambda, T)] = \max \left\{ \frac{\lambda 2^{k_\lambda((1-\delta)y)-1}}{1-\alpha} (2(1-\alpha) - q_\lambda), \lambda 2^{k_\lambda((1-\delta)y)-1} \right\},$$

which concludes the proof. ■

Table 5.1: Experimental results for ski rental algorithms.

	MAX	AVG	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + br/2$	$b(r - 1)$
Avg perf. ratio	1.344	1.337	1.340	1.349	1.367	1.677	2.187	2.219
CI$_{+}$/CI$_{-}$	+0.03/-0.03	+0.03/-0.03	+0.03/-0.03	+0.03/-0.03	+0.03/-0.04	+0.05/-0.05	+0.16/-0.17	+0.17/-0.17
Exp. cost	11.241	11.187	11.173	11.215	11.316	16.987	21.234	20.958
CI$_{+}$/CI$_{-}$	+0.48/-0.51	+0.49/-0.51	+0.50/-0.52	+0.48/-0.54	+0.48/-0.52	+0.99/-1.09	+2.01/-2.22	+1.95/-2.00

5.6 Experimental Evaluation

We analyze experimentally our algorithms of Section 5.3, Section 5.5 and Section 5.4 which optimize the maximum and average distance as well as the CVar. We refer to these algorithms as MAX, AVG and CVar $_{\alpha}$.

5.6.1 Evaluation of Ski Rental Algorithms

Baselines We compare our algorithms to the class BP $_{\rho}$ introduced in [BP25]. The algorithm BP $_{\rho}$ buys at time $b/(r - 1)$ if $y \geq b$, and at time $\rho \in [b, b(r - 1)]$ otherwise. In our experiments we use $\rho \in \{b, b + \frac{br}{2}, b(r - 1)\}$.

Datasets We set $b = 10$ and $r = 5$. The prediction y is drawn from $\text{Unif}[b/z, bz]$ with $z = 4$, and the prediction range is defined as $R_y = [(1 - \delta)y, (1 + \delta)y]$ with $\delta = 0.9$. The true horizon x is selected uniformly at random from R_y .

Evaluation We measure two quantities: (i) the average performance ratio over all $x \in R_y$, and (ii) the expected cost under the distribution μ over R_y . Both quantities are averaged over 1000 independent draws of y . For MAX and AVG, we use a symmetric linear weight function over R_y . For CVar $_{\alpha}$, the distribution μ follows a truncated Gaussian centered at y , with $\alpha \in \{0.1, 0.5, 0.9\}$. Table 5.1 reports the average performance ratios, expected costs, and their 95% confidence intervals.

Extended results We also conducted experiments with varying parameters (δ, r, z) , and with two types of weight functions on R_y : (i) a linear symmetric weight decreasing from 1 at y to 0 at the endpoints, and (ii) a Gaussian weight with mean y and standard deviation $\sigma = \delta y/4$, both truncated and normalized. The same evaluation protocol was applied to MAX, AVG, and CVar $_{\alpha}$, with $\alpha \in \{0.1, 0.5, 0.9\}$. Tables 5.2–5.11 summarize the results for the five parameter settings $(\delta, r, z) \in \{(0.9, 5, 4), (0.9, 8, 4), (0.5, 5, 4), (0.5, 8, 4), (0.9, 5, 7)\}$.

Discussion Across all settings, our algorithms consistently outperform BP $_{\rho}$ in both performance ratio and expected cost. The difference becomes larger for higher uncertainty (larger δ), where our algorithms adapt more effectively to the

Table 5.2: Ski rental (linear weight), $\delta = 0.9$, $r = 5$, $z = 4$.

	MAX	AVG	CVAR $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.3442	1.3368	1.3418	1.3637	1.3930	1.6767	2.1874	2.2189
CI$_{+}$/CI$_{-}$	+0.0311/- 0.0345	+0.0309/- 0.0336	+0.0297/- 0.0303	+0.0343/- 0.0342	+0.0366/- 0.0355	+0.0482/- 0.0502	+0.1606/- 0.1668	+0.1706/- 0.1684
Exp. cost	11.2501	11.1832	11.1693	11.2758	11.4553	16.4188	20.6070	20.5024
CI$_{+}$/CI$_{-}$	+0.4798/- 0.5076	+0.4866/- 0.5085	+0.4954/- 0.5201	+0.4787/- 0.5368	+0.4671/- 0.4981	+0.9002/- 0.9968	+1.8565/- 2.0281	+1.8461/- 1.8712

Table 5.3: Ski rental (linear weight), $\delta = 0.9$, $r = 8$, $z = 4$.

	MAX	AVG	CVAR $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.2879	1.2853	1.3031	1.3343	1.3978	1.6767	2.2417	2.2251
CI$_{+}$/CI$_{-}$	+0.0318/- 0.0339	+0.0324/- 0.0334	+0.0338/- 0.0355	+0.0435/- 0.0398	+0.0543/- 0.0505	+0.0482/- 0.0502	+0.1794/- 0.1831	+0.1748/- 0.1732
Exp. cost	10.4306	10.3880	10.3999	10.4840	10.7296	16.4325	20.8787	20.7540
CI$_{+}$/CI$_{-}$	+0.3959/- 0.4364	+0.4165/- 0.4321	+0.4189/- 0.4595	+0.4035/- 0.4551	+0.3811/- 0.4256	+0.9005/- 0.9964	+1.9688/- 2.1559	+1.9242/- 1.9548

Table 5.4: Ski rental (linear weight), $\delta = 0.5$, $r = 5$, $z = 4$.

	MAX	AVG	CVAR $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.2047	1.2034	1.2085	1.2145	1.2287	1.7729	2.2041	2.1976
CI$_{+}$/CI$_{-}$	+0.0182/- 0.0201	+0.0182/- 0.0204	+0.0170/- 0.0176	+0.0182/- 0.0198	+0.0216/- 0.0223	+0.0633/- 0.0695	+0.1836/- 0.1871	+0.1870/- 0.1844
Exp. cost	11.1739	11.1732	11.1751	11.2170	11.3174	17.0456	21.2944	21.0003
CI$_{+}$/CI$_{-}$	+0.4695/- 0.5043	+0.4829/- 0.5080	+0.4960/- 0.5209	+0.4791/- 0.5392	+0.4821/- 0.5183	+0.9977/- 1.1034	+2.0426/- 2.2322	+1.9736/- 2.0155

prediction range. As δ decreases, the performance ratio approaches 1, reflecting better alignment between the predicted and actual horizons. The influence of r and z is limited, confirming that the algorithms remain stable with respect to these parameters. Finally, Gaussian weighting tends to yield slightly better results than linear weighting, since it emphasizes outcomes near the predicted value y more strongly.

Table 5.5: Ski rental (linear weight), $\delta = 0.5$, $r = 8$, $z = 4$.

	Max	Avg	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.1265	1.1245	1.1334	1.1404	1.1476	1.7729	2.1680	2.1595
CI$_{+}$/CI$_{-}$	+0.0112/- 0.0125	+0.0108/- 0.0124	+0.0098/- 0.0098	+0.0133/- 0.0128	+0.0158/- 0.0153	+0.0633/- 0.0695	+0.1825/- 0.1828	+0.1783/- 0.1757
Exp. cost	10.3897	10.3880	10.3983	10.4419	10.4889	17.0499	20.7854	20.7519
CI$_{+}$/CI$_{-}$	+0.3918/- 0.4351	+0.4165/- 0.4321	+0.4184/- 0.4550	+0.3959/- 0.4595	+0.4063/- 0.4330	+0.9967/- 1.1027	+1.9364/- 2.1260	+1.9246/- 1.9540

Table 5.6: Ski rental (linear weight), $\delta = 0.9$, $r = 5$, $z = 7$.

	Max	Avg	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.3103	1.3053	1.3151	1.3295	1.3534	1.7901	2.8509	2.9905
CI$_{+}$/CI$_{-}$	+0.0229/- 0.0222	+0.0233/- 0.0223	+0.0238/- 0.0260	+0.0276/- 0.0275	+0.0312/- 0.0310	+0.0456/- 0.0472	+0.1914/- 0.2102	+0.2160/- 0.2153
Exp. cost	11.7081	11.7023	11.7078	11.7775	11.9247	17.8170	27.2827	27.5432
CI$_{+}$/CI$_{-}$	+0.3917/- 0.4533	+0.4131/- 0.4432	+0.4053/- 0.4532	+0.3772/- 0.4741	+0.3763/- 0.4357	+0.7726/- 0.8356	+2.0175/- 2.3302	+2.0996/- 2.2661

Table 5.7: Ski rental (Gaussian weight), $\delta = 0.9$, $r = 5$, $z = 4$.

	Max	Avg	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.3514	1.3387	1.3397	1.3492	1.3669	1.6767	2.1874	2.2189
CI$_{+}$/CI$_{-}$	+0.0297/- 0.0333	+0.0307/- 0.0337	+0.0302/- 0.0311	+0.0326/- 0.0344	+0.0335/- 0.0358	+0.0482/- 0.0502	+0.1606/- 0.1668	+0.1706/- 0.1684
Exp. cost	11.1765	11.1731	11.1731	11.2151	11.3155	16.9873	21.2338	20.9577
CI$_{+}$/CI$_{-}$	+0.4687/- 0.5039	+0.4829/- 0.5081	+0.4966/- 0.5217	+0.4796/- 0.5398	+0.4828/- 0.5188	+0.9850/- 1.0909	+2.0143/- 2.2246	+1.9520/- 1.9971

5.6.2 Evaluation of 1-Max Search Algorithms

Baselines We compare our algorithms to three reference baselines. The first is the Pareto-optimal algorithm of [Sun+21], denoted by PO $_1$. The second, denoted by PO $_2$ [AK23b], sets its threshold to $T = \min\{t_2, \max\{t_1, y\}\}$, where $t_1 = M/r$ and $t_2 = r$. We also include the δ -tolerant algorithm of [AKZ22], denoted by δ -TOL.

Datasets We set $M = 1000$ and $r = 100$. The prediction y is drawn from $\text{Unif}[z, M/z]$ with $z = 10$, and the prediction range is $R_y = [(1 - \delta)y, (1 + \delta)y]$, where $\delta = 0.9$. The input is the standard worst-case sequence of increasing prices

Table 5.8: Ski rental (Gaussian weight), $\delta = 0.5$, $r = 5$, $z = 4$.

	MAX	AVG	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.2097	1.2034	1.2049	1.2049	1.2141	1.7729	2.2041	2.1976
CI$_{+}$/CI$_{-}$	+0.0184/- 0.0206	+0.0182/- 0.0204	+0.0180/- 0.0185	+0.0185/- 0.0199	+0.0197/- 0.0221	+0.0633/- 0.0695	+0.1836/- 0.1871	+0.1870/- 0.1844
Exp. cost	11.1740	11.1732	11.1733	11.1733	11.2349	17.1177	21.3451	20.9444
CI$_{+}$/CI$_{-}$	+0.4693/- 0.5043	+0.4829/- 0.5080	+0.4966/- 0.5217	+0.4843/- 0.5294	+0.4917/- 0.5055	+1.0257/- 1.1153	+2.0850/- 2.2479	+1.9824/- 2.0130

Table 5.9: Ski rental (Gaussian weight), $\delta = 0.9$, $r = 8$, $z = 4$.

	MAX	AVG	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.3025	1.2853	1.3002	1.3175	1.3571	1.6767	2.2417	2.2251
CI$_{+}$/CI$_{-}$	+0.0322/- 0.0335	+0.0324/- 0.0334	+0.0342/- 0.0360	+0.0416/- 0.0390	+0.0507/- 0.0444	+0.0482/- 0.0502	+0.1794/- 0.1831	+0.1748/- 0.1732
Exp. cost	10.3947	10.3880	10.3983	10.4381	10.5760	16.9918	20.7954	20.7520
CI$_{+}$/CI$_{-}$	+0.3933/- 0.4372	+0.4165/- 0.4321	+0.4198/- 0.4603	+0.3974/- 0.4614	+0.4016/- 0.4385	+0.9841/- 1.0901	+1.9368/- 2.1269	+1.9246/- 1.9540

Table 5.10: Ski rental (Gaussian weight), $\delta = 0.5$, $r = 8$, $z = 4$.

	MAX	AVG	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.1345	1.1245	1.1288	1.1324	1.1404	1.7729	2.1680	2.1595
CI$_{+}$/CI$_{-}$	+0.0147/- 0.0149	+0.0108/- 0.0124	+0.0103/- 0.0105	+0.0123/- 0.0122	+0.0152/- 0.0150	+0.0633/- 0.0695	+0.1825/- 0.1828	+0.1783/- 0.1757
Exp. cost	10.3908	10.3880	10.3898	10.4073	10.4565	17.1216	20.7559	20.7519
CI$_{+}$/CI$_{-}$	+0.3920/- 0.4354	+0.4165/- 0.4321	+0.4204/- 0.4565	+0.3986/- 0.4622	+0.4107/- 0.4390	+1.0255/- 1.1160	+1.9374/- 2.1215	+1.9246/- 1.9540

up to the maximum price x , followed by a last price equal to 1, where x is selected uniformly at random from R_y [Ele+24; Sun+21].

Evaluation We compute two quantities: (i) the average performance ratio over all $x \in R_y$, and (ii) the expected profit under a distribution μ over R_y . Both are averaged over 1000 independent draws of y . For MAX and AVG, we use a symmetric linear weight function over R_y . For CVar $_{\alpha}$, the distribution μ follows a truncated Gaussian centered at y , with $\alpha \in \{0.1, 0.5, 0.9\}$. Table 5.12 reports the average performance ratios, expected profits, and 95% confidence intervals.

Extended results We also tested the algorithms for different values of the

Table 5.11: Ski rental (Gaussian weight), $\delta = 0.9$, $r = 5$, $z = 7$.

	Max	Avg	CVar $_{\alpha}$			BP $_{\rho}$		
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	b	$b + \frac{br}{2}$	$b(r - 1)$
Avg perf. ratio	1.3201	1.3065	1.3110	1.3199	1.3383	1.7901	2.8509	2.9905
CI$_{+}$/CI$_{-}$	+0.0244/- 0.0231	+0.0236/- 0.0227	+0.0238/- 0.0262	+0.0262/- 0.0258	+0.0313/- 0.0288	+0.0456/- 0.0472	+0.1914/- 0.2102	+0.2160/- 0.2153
Exp. cost	11.7063	11.7000	11.7051	11.7411	11.8475	18.2168	30.3389	30.9557
CI$_{+}$/CI$_{-}$	+0.3918/- 0.4544	+0.4139/- 0.4431	+0.4048/- 0.4595	+0.3831/- 0.4670	+0.4039/- 0.4368	+0.7890/- 0.8573	+2.4735/- 2.9018	+2.6507/- 2.7672

Table 5.12: Experimental results for 1-Max search algorithms.

	Max	Avg	CVar $_{\alpha}$			δ -ToL	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	4.394	4.447	9.771	8.144	6.022	10.009	4.630	15.685
CI$_{+}$/CI$_{-}$	+0.04/-0.05	+0.06/-0.05	+0.26/-0.26	+0.20/-0.20	+0.12/-0.12	+0.00/-0.00	+0.06/-0.06	+0.45/-0.45
Exp. profit	15.614	20.528	35.795	34.402	27.500	5.475	13.904	27.986
CI$_{+}$/CI$_{-}$	+0.30/-0.32	+0.50/-0.48	+1.06/-1.04	+1.00/-1.08	+0.81/-0.82	+0.15/-0.17	+0.16/-0.17	+0.82/-0.80

parameters (δ, r, z) and for two types of weight functions on R_y : (i) a symmetric linear weight decreasing from 1 at y to 0 at the endpoints, and (ii) a Gaussian weight with mean y and standard deviation $\sigma = \delta y/4$, both truncated and normalized. The same evaluation protocol was applied to MAX, AVG, and CVar $_{\alpha}$ with $\alpha \in \{0.1, 0.5, 0.9\}$. Tables 5.13–5.22 summarize the results for the five parameter settings $(\delta, r, z) \in \{(0.9, 100, 10), (0.5, 100, 10), (0.9, 80, 10), (0.5, 80, 10), (0.9, 100, 20)\}$.

Discussion Overall, our algorithms achieve better performance ratios and expected profits than all baselines. Among the baselines, PO $_2$ tends to be the most brittle, since it sets a high threshold, while δ -ToL is the most conservative. PO $_1$ achieves a balance between the two but still lags behind our proposed methods. MAX and AVG consistently outperform the baselines across both metrics, with AVG slightly ahead in terms of expected profit. For the CVar class, the expected profit decreases with α , while the performance ratio improves, confirming the trade-off between robustness and risk. When varying δ , smaller values lead to better performance, as the prediction range narrows and uncertainty decreases. The influence of r and z remains small, showing that the algorithms are stable across different scales. Finally, Gaussian weighting tends to provide slightly better results than linear weighting, as it focuses more strongly on outcomes near the predicted value y .

Table 5.13: 1-Max (linear weight), $\delta = 0.9$, $r = 100$, $z = 10$.

	MAX	AVG	CVAR $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	4.3942	4.4469	9.9327	6.7335	5.2429	10.0085	4.6304	15.6848
CI$_+$/CI$_-$	+0.04/-0.05	+0.06/-0.05	+0.26/-0.26	+0.15/-0.15	+0.12/-0.11	+0.00/-0.00	+0.06/-0.06	+0.45/-0.45
Exp. profit	15.2276	19.6283	30.7586	27.7794	16.1624	5.4746	13.5565	27.9437
CI$_+$/CI$_-$	+0.30/-0.33	+0.49/-0.48	+0.90/-0.89	+0.80/-0.87	+0.49/-0.50	+0.15/-0.17	+0.18/-0.18	+0.82/-0.80

Table 5.14: 1-Max (linear weight), $\delta = 0.5$, $r = 100$, $z = 10$.

	MAX	AVG	CVAR $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	2.5877	2.7856	10.0269	6.7880	3.1589	2.0000	3.8718	21.0408
CI$_+$/CI$_-$	+0.02/-0.02	+0.01/-0.01	+0.26/-0.26	+0.16/-0.16	+0.04/-0.04	+0.00/-0.00	+0.06/-0.07	+0.62/-0.62
Exp. profit	28.6754	29.2159	36.1414	34.5976	29.7982	27.3730	13.9343	28.0349
CI$_+$/CI$_-$	+0.79/-0.88	+0.86/-0.82	+1.07/-1.05	+1.01/-1.09	+0.86/-0.87	+0.75/-0.83	+0.16/-0.16	+0.82/-0.81

Table 5.15: 1-Max (linear weight), $\delta = 0.9$, $r = 80$, $z = 10$.

	MAX	AVG	CVAR $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	4.4865	4.4469	9.9943	6.9624	5.7321	10.0085	4.6516	15.6848
CI$_+$/CI$_-$	+0.04/-0.04	+0.06/-0.05	+0.26/-0.26	+0.15/-0.15	+0.14/-0.14	+0.00/-0.00	+0.05/-0.05	+0.45/-0.45
Exp. profit	15.6165	19.6283	30.7121	27.6555	15.8462	5.4746	14.4844	27.9437
CI$_+$/CI$_-$	+0.28/-0.31	+0.49/-0.48	+0.91/-0.89	+0.81/-0.88	+0.52/-0.53	+0.15/-0.17	+0.17/-0.18	+0.82/-0.80

Table 5.16: 1-Max (linear weight), $\delta = 0.5$, $r = 80$, $z = 10$.

	MAX	AVG	CVAR $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	2.7301	2.7856	10.0378	6.7586	3.1362	2.0000	3.8570	21.0408
CI$_+$/CI$_-$	+0.05/-0.05	+0.01/-0.01	+0.26/-0.26	+0.16/-0.16	+0.04/-0.04	+0.00/-0.00	+0.06/-0.07	+0.62/-0.62
Exp. profit	28.6082	29.2159	36.0559	34.5122	29.7079	27.3730	14.9631	28.0349
CI$_+$/CI$_-$	+0.78/-0.87	+0.86/-0.82	+1.07/-1.04	+1.02/-1.10	+0.86/-0.87	+0.75/-0.83	+0.16/-0.17	+0.82/-0.81

Real Data Experiments for 1-Max Search

In this section, we provide a computational evaluation of our algorithms on real-world data, using the same algorithm baselines as above. We consider two datasets:

Table 5.17: 1-Max (linear weight), $\delta = 0.9$, $r = 100$, $z = 20$.

	Max	Avg	CVar $_{\alpha}$			δ -ToL	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	3.8763	3.8645	6.7675	4.8622	4.1063	10.0049	3.8668	10.2558
CI$_+$/CI$_-$	+0.02/-0.02	+0.02/-0.02	+0.09/-0.09	+0.05/-0.05	+0.07/-0.07	+0.00/-0.00	+0.02/-0.02	+0.15/-0.15
Exp. profit	11.2099	13.9404	19.7256	17.8093	10.9724	3.4915	11.7401	18.0238
CI$_+$/CI$_-$	+0.10/-0.10	+0.18/-0.17	+0.30/-0.30	+0.27/-0.29	+0.15/-0.15	+0.05/-0.06	+0.06/-0.06	+0.28/-0.27

Table 5.18: 1-Max (Gaussian weight), $\delta = 0.9$, $r = 100$, $z = 10$.

	Max	Avg	CVar $_{\alpha}$			δ -ToL	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	5.0599	5.4540	9.7709	8.1444	6.0222	10.0085	4.6304	15.6848
CI$_+$/CI$_-$	+0.0728/- 0.0756	+0.0940/- 0.0909	+0.2564/- 0.2561	+0.1988/- 0.1969	+0.1196/- 0.1205	+0.0008/- 0.0008	+0.0561/- 0.0578	+0.4526/- 0.4530
Exp. profit	24.9584	27.0416	35.7945	34.4015	27.5003	5.4746	13.9039	27.9858
CI$_+$/CI$_-$	+0.6259/- 0.7080	+0.7406/- 0.7242	+1.0580/- 1.0403	+1.0003/- 1.0822	+0.8137/- 0.8159	+0.1503/- 0.1666	+0.1583/- 0.1659	+0.8233/- 0.8022

Table 5.19: 1-Max (Gaussian weight), $\delta = 0.5$, $r = 100$, $z = 10$.

	Max	Avg	CVar $_{\alpha}$			δ -ToL	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	5.9298	6.1761	12.1837	10.5279	7.4024	2.0000	3.8718	21.0408
CI$_+$/CI$_-$	+0.1000/- 0.1051	+0.1142/- 0.1099	+0.3366/- 0.3318	+0.2822/- 0.2819	+0.1767/- 0.1844	+0.0000/- 0.0000	+0.0622/- 0.0653	+0.6202/- 0.6154
Exp. profit	35.2058	35.6679	41.5955	40.8581	37.3962	27.3730	13.9997	28.1314
CI$_+$/CI$_-$	+0.9612/- 1.0665	+1.0397/- 1.0025	+1.2372/- 1.2094	+1.1976/- 1.2908	+1.0807/- 1.0969	+0.7512/- 0.8332	+0.1485/- 0.1558	+0.8152/- 0.8127

Table 5.20: 1-Max (Gaussian weight), $\delta = 0.9$, $r = 80$, $z = 10$.

	Max	Avg	CVar $_{\alpha}$			δ -ToL	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	5.1262	5.4540	9.8288	8.3568	6.2563	10.0085	4.6516	15.6848
CI$_+$/CI$_-$	+0.0622/- 0.0690	+0.0940/- 0.0909	+0.2528/- 0.2506	+0.1908/- 0.1933	+0.1229/- 0.1208	+0.0008/- 0.0008	+0.0465/- 0.0473	+0.4526/- 0.4530
Exp. profit	24.8628	27.0416	35.6981	34.2272	27.3657	5.4746	14.9229	27.9858
CI$_+$/CI$_-$	+0.6067/- 0.6877	+0.7406/- 0.7242	+1.0671/- 1.0390	+1.0189/- 1.0926	+0.8320/- 0.8334	+0.1503/- 0.1666	+0.1671/- 0.1722	+0.8233/- 0.8022

Table 5.21: 1-Max (Gaussian weight), $\delta = 0.5$, $r = 80$, $z = 10$.

	MAX	AVG	CVar $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	5.9483	6.1761	12.1798	10.5404	7.3715	2.0000	3.8570	21.0408
CI$_+$/CI$_-$	+0.0807/- 0.0907	+0.1142/- 0.1099	+0.3329/- 0.3376	+0.2871/- 0.2838	+0.1781/- 0.1906	+0.0000/- 0.0000	+0.0629/- 0.0660	+0.6202/- 0.6154
Exp. profit	34.9084	35.6679	41.4923	40.7649	37.2693	27.3730	15.0337	28.1314
CI$_+$/CI$_-$	+0.9659/- 1.0843	+1.0397/- 1.0025	+1.2443/- 1.2156	+1.1989/- 1.3073	+1.0851/- 1.1117	+0.7512/- 0.8332	+0.1612/- 0.1756	+0.8152/- 0.8127

Table 5.22: 1-Max (Gaussian weight), $\delta = 0.9$, $r = 100$, $z = 20$.

	MAX	AVG	CVar $_{\alpha}$			δ -Tol	PO $_1$	PO $_2$
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
Avg perf. ratio	4.1276	4.3758	6.6495	5.6691	4.2717	10.0049	3.8668	10.2558
CI$_+$/CI$_-$	+0.0295/- 0.0313	+0.0344/- 0.0339	+0.0853/- 0.0854	+0.0664/- 0.0676	+0.0365/- 0.0386	+0.0004/- 0.0004	+0.0217/- 0.0227	+0.1520/- 0.1489
Exp. profit	16.9932	18.3253	22.8969	21.9870	17.7082	3.4915	12.2219	18.0633
CI$_+$/CI$_-$	+0.2258/- 0.2519	+0.2575/- 0.2496	+0.3510/- 0.3484	+0.3316/- 0.3609	+0.2615/- 0.2685	+0.0501/- 0.0555	+0.0495/- 0.0505	+0.2777/- 0.2677

(i) the exchange rates⁴ of EUR to four other currencies (CHF, USD, JPY, and GBP), where each series is a sequence σ of 6672 daily prices over a span of 25 years; and (ii) Bitcoin (USD) data recorded every minute from January 1st 2020 to December 31st 2024, comprising a total of 2,630,880 prices,⁵ following [Sun+21] and [Ben+25].

Datasets For each sequence σ , let $x = \max_t \sigma_t$ denote the maximum price in the input. For generating predictions, we consider a random value z sampled from a normal distribution with mean 0, standard deviation $1/2$, truncated to $[-1, +1]$. This is scaled by the error upper bound δ to obtain $y = x + x\delta \cdot z$. The error bound δ is obtained by partitioning σ into eight equal-length segments, letting M_i be the maximum in segment i , and setting $x\delta = \max_i M_i - \min_i M_i$. In 1-max search, if all prices are below the chosen threshold, the algorithm sells at the lowest price in the sequence.

Evaluation We performed 10,000 runs and report average performance ratio and expected profit with 95% CIs. For MAX and AVG we use linear symmetric weights; CVar $_{\alpha}$ uses a Gaussian truncated to R_y , with $\alpha \in \{0.1, 0.5, 0.9\}$.

⁴ https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html

⁵ <https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data?resource=download>

Table 5.23: Real-data evaluation for 1-max search: average performance ratios and expected profits with 95% confidence intervals.

Currency	MAX	Avg	CVar $_{\alpha}$			δ -ToL	PO ₁	PO ₂
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$			
CHF (Avg. ratio)	1.2617	1.2503	1.3287	1.3178	1.3451	1.5286	1.7824	1.4702
CI₊/CI₋	+0.013/- 0.014	+0.016/- 0.015	+0.019/- 0.018	+0.016/- 0.015	+0.015/- 0.014	+0.015/- 0.014	+0.002/- 0.002	+0.021/- 0.019
CHF (Exp. profit)	1.612	1.624	1.587	1.553	1.498	1.372	1.298	1.462
CI₊/CI₋	+0.031/- 0.028	+0.036/- 0.032	+0.027/- 0.029	+0.028/- 0.027	+0.026/- 0.025	+0.021/- 0.020	+0.019/- 0.018	+0.030/- 0.028
GBP (Avg. ratio)	1.1573	1.1573	1.1342	1.1137	1.0912	1.1474	1.1573	1.1241
CI₊/CI₋	+0.002/- 0.001	+0.001/- 0.001	+0.004/- 0.003	+0.004/- 0.003	+0.004/- 0.003	+0.002/- 0.002	+0.001/- 0.001	+0.004/- 0.003
GBP (Exp. profit)	0.927	0.918	0.944	0.931	0.912	0.856	0.807	0.884
CI₊/CI₋	+0.014/- 0.013	+0.012/- 0.011	+0.018/- 0.016	+0.017/- 0.015	+0.016/- 0.015	+0.014/- 0.013	+0.010/- 0.009	+0.018/- 0.016
JPY (Avg. ratio)	1.0842	1.0842	1.0842	1.0842	1.0842	1.0842	1.0876	1.0741
CI₊/CI₋	+0.001/- 0.001	+0.001/- 0.001	+0.001/- 0.001	+0.001/- 0.001	+0.001/- 0.001	+0.001/- 0.001	+0.001/- 0.001	+0.002/- 0.001
JPY (Exp. profit)	168.4	168.3	168.7	168.5	168.0	167.2	166.8	169.1
CI₊/CI₋	+1.5/-1.4	+1.6/-1.5	+1.6/-1.5	+1.5/-1.6	+1.6/-1.5	+1.3/-1.3	+1.2/-1.2	+2.0/-1.9
USD (Avg. ratio)	1.2042	1.1837	1.2289	1.2254	1.2471	1.3582	1.5439	1.3263
CI₊/CI₋	+0.011/- 0.010	+0.010/- 0.009	+0.012/- 0.011	+0.012/- 0.011	+0.011/- 0.010	+0.011/- 0.010	+0.001/- 0.001	+0.015/- 0.013
USD (Exp. profit)	1.451	1.477	1.503	1.469	1.392	1.327	1.213	1.424
CI₊/CI₋	+0.023/- 0.022	+0.025/- 0.023	+0.029/- 0.027	+0.027/- 0.028	+0.028/- 0.026	+0.021/- 0.020	+0.012/- 0.012	+0.031/- 0.029
BTC (Avg. ratio)	9.0380	8.8881	9.0380	9.0380	9.0380	9.0380	15.1874	9.4486
CI₊/CI₋	+0.38/-0.36	+0.37/-0.36	+0.38/-0.37	+0.39/-0.38	+0.39/-0.37	+0.37/-0.36	+0.02/-0.02	+0.42/-0.41
BTC (Exp. profit)	24,132	24,228	24,180	24,095	23,978	23,842	23,610	24,310
CI₊/CI₋	+812/-796	+824/-781	+897/-873	+852/-829	+783/-764	+653/-641	+514/-487	+947/-932

Results Table 5.23 reports averages and CIs. Since real sequences are not worst-case and price ranges vary by asset, performance varies across datasets; nonetheless, MAX, AVG and CVar_{0.5} are overall competitive with or better than the baselines. Table 5.24 lists price ranges to contextualize variability (BTC shows a ratio exceeding 28).

5.6.3 Evaluation of Contract Schedules

Baselines We compare our schedules against two 4-robust baselines [AK23a]: the Pareto-optimal schedule (PO), which completes a contract at the prediction y , and the δ -tolerant schedule (δ -ToL), which completes a contract at $(1 - \delta)y$.

Datasets The prediction is chosen as $y \sim \text{Unif}[0.8 \cdot 10^6, 1.2 \cdot 10^6]$, and the

Table 5.24: Lowest and highest prices observed in each currency sequence, and their ratio.

Currency	Lowest	Highest	Ratio
CHF	0.9260	1.6803	1.8146
GBP	0.5711	0.9786	1.7134
JPY	89.3000	175.3900	1.9641
USD	0.8252	1.5990	1.9377
BTC	3865.0	108276.0	28.0145

prediction range is $R_y = [(1 - \delta)y, (1 + \delta)y]$. The interruption time T is chosen uniformly at random from R_y .

Evaluation For MAX and AVG, we use two weight functions defined on R_y . The first is a linear symmetric function that decreases from 1 at y to 0 at the endpoints of R_y . The second is a Gaussian function

$$w(x) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-y}{\sigma}\right)^2\right), & \text{if } x \in R_y, \\ 0, & \text{otherwise,} \end{cases}$$

where $\sigma = \delta y/4$. For CVAR_α , the predictive distribution μ coincides with the respective weight function, truncated and normalized on R_y , with $\alpha \in \{0.1, 0.5, 0.9\}$.

For each schedule, we compute the average performance ratio, taken over all $T \in R_y$, with 1000 independent repetitions on the choice of y . We also compute the expected completed contract length under μ , with the averaging performed over the choices of y . The tables also report the 95% confidence intervals (CIs). Results are presented in six tables: three for the linear weight function (Tables 5.25–5.27) and three for the Gaussian weight function (Tables 5.28–5.30), each corresponding to a different value of δ .

Discussion The tables show that, in the vast majority of the considered settings, our schedules achieve better performance ratios and larger expected contract lengths than both PO and δ -Tol. This can be explained by the fact that all Pareto-optimal algorithms are brittle, as shown in [Ele+24], whereas, in contrast, the δ -Tol algorithm is inefficient unless the prediction error is large.

For CVAR_α , the results show that the expected completed contract length decreases as α grows, while the performance ratio tends to increase, which is consistent with the tradeoff between risk and robustness. When δ is smaller, all schedules perform better: this is because the reduced prediction range allows our algorithms a more accurate positioning of the completion time. In a similar vein, for Gaussian weights, the results are consistently stronger than for linear weights, because the distribution is more concentrated around y . Finally, we note that most confidence

Table 5.25: Experimental results for contract scheduling (linear weight), $\delta = 0.2$.

	MAX	Avg	CVAR $_{\alpha}$			PO	δ -TOL
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.421	2.413	2.426	2.416	2.471	2.960	2.500
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0000/- 0.0000	+0.0002/- 0.0002	+0.0001/- 0.0001	+0.0002/- 0.0003	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	413,303	417,504	420,164	415,274	402,646	373,008	397,875
CI$_{+}$/CI$_{-}$	+9083.55/- 8957.96	+8495.21/- 9051.60	+9123.35/- 8027.55	+8621.83/- 8983.43	+8679.61/- 8684.86	+7776.46/- 7024.64	+8377.65/- 8357.20

Table 5.26: Experimental results for contract scheduling (linear weight), $\delta = \frac{1}{3}$.

	MAX	Avg	CVAR $_{\alpha}$			PO	δ -TOL
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.689	2.632	2.643	2.654	2.873	2.934	3.000
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0000/- 0.0000	+0.0002/- 0.0002	+0.0002/- 0.0002	+0.0006/- 0.0006	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	371,190	387,025	389,927	378,469	344,100	370,414	329,257
CI$_{+}$/CI$_{-}$	+7982.66/- 7670.94	+8086.65/- 8892.81	+8739.76/- 9252.04	+8779.84/- 7975.69	+7710.75/- 8106.44	+8252.32/- 8220.64	+7185.13/- 7391.59

Table 5.27: Experimental results for contract scheduling (linear weight), $\delta = 0.4$.

	MAX	Avg	CVAR $_{\alpha}$			PO	δ -TOL
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.808	2.704	2.711	2.746	3.031	2.920	3.287
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0000/- 0.0000	+0.0001/- 0.0001	+0.0003/- 0.0003	+0.0003/- 0.0003	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	361,661	385,593	387,891	372,668	335,125	376,514	308,199
CI$_{+}$/CI$_{-}$	+7657.21/- 7979.53	+8265.97/- 8367.98	+8415.05/- 8767.30	+8449.57/- 8393.17	+7200.61/- 7299.36	+8415.63/- 8840.35	+7370.45/- 8120.64

intervals on the reported objectives collapse to zero. This is consistent with theory because the schedules have the same structure for each prediction y .

5.7 Conclusion

We introduced new evaluation metrics inspired by decision theory that enable the optimization of learning-augmented algorithms over the entire range of prediction error. Using these metrics, we designed theoretically optimal algorithms for three

Table 5.28: Experimental results for contract scheduling (Gaussian weight), $\delta = 0.2$.

	MAX	Avg	CVar $_{\alpha}$			PO	δ -Tol
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.267	2.266	2.273	2.266	2.315	2.960	2.500
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0001/- 0.0001	+0.0002/- 0.0002	+0.0000/- 0.0001	+0.0003/- 0.0003	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	442,104	443,434	445,046	442,662	430,335	373,008	397,869
CI$_{+}$/CI$_{-}$	+9722.63/- 9586.90	+9022.41/- 9592.05	+9673.17/- 8505.70	+9198.55/- 9603.26	+9289.41/- 9245.24	+7776.46/- 7024.64	+8377.52/- 8357.07

Table 5.29: Experimental results for contract scheduling (Gaussian weight), $\delta = \frac{1}{3}$.

	MAX	Avg	CVar $_{\alpha}$			PO	δ -Tol
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.421	2.413	2.423	2.415	2.524	2.934	3.000
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0001/- 0.0001	+0.0002/- 0.0002	+0.0001/- 0.0001	+0.0004/- 0.0004	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	412,448	416,801	419,095	414,520	392,171	370,414	329,262
CI$_{+}$/CI$_{-}$	+8872.45/- 8524.03	+8664.99/- 9576.50	+9395.10/- 9951.62	+9623.35/- 8751.30	+8729.12/- 9156.69	+8252.32/- 8220.64	+7185.24/- 7391.71

Table 5.30: Experimental results for contract scheduling (Gaussian weight), $\delta = 0.4$.

	MAX	Avg	CVar $_{\alpha}$			PO	δ -Tol
			$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$		
Avg perf. ratio	2.494	2.478	2.488	2.483	2.631	2.920	3.287
CI$_{+}$/CI$_{-}$	+0.0000/- 0.0000	+0.0001/- 0.0001	+0.0003/- 0.0002	+0.0002/- 0.0002	+0.0006/- 0.0006	+0.0000/- 0.0000	+0.0000/- 0.0000
Exp. contract length	407,466	413,970	416,511	410,594	382,442	376,514	308,032
CI$_{+}$/CI$_{-}$	+8625.69/- 8993.79	+8887.56/- 8941.28	+9024.70/- 9400.35	+9319.17/- 9231.67	+8202.65/- 8300.45	+8415.63/- 8840.35	+6701.11/- 7383.17

well-studied problems and demonstrated that they outperform the known, extreme case approaches in practice.

Our techniques have the potential to extend to other learning-augmented problems, such as knapsack [Dan+24] and secretary problems [Ant+23], where single-valued ML-predictions have already shown to be effective. Another direction for future work is problems with multi-valued predictions such as packing problems [Im+21]. Since prediction error in these cases can also be quantified through a norm between the predicted and actual vectors, our framework remains applicable and could offer meaningful performance guarantees in more complex settings.

This thesis explored algorithmic models and techniques for sequential decision-making under uncertainty. The primary focus was on two approaches: scenario-based robust optimization and learning-augmented analysis. We developed formal frameworks for each approach, provided algorithmic results supported by theoretical guarantees, and complemented them with experimental evaluations.

In the first part of the thesis, we introduced a scenario-based model for fundamental data structures. More precisely, in Chapter 3, we presented the novel study of binary search trees and Huffman trees in a robust setting where the input is drawn from an unknown scenario among a finite set. We analyzed the computational complexity of these problems, proving that constructing optimal trees is NP-hard, and proposed approximation algorithms with provable performance bounds. Our results use the competitive ratio and the regret as performance metrics, establishing tight bounds in both cases. In addition, we proposed a fairness-based interpretation of regret, offering a new perspective on how to balance performance across multiple users. This part also included an experimental evaluation that supports the theoretical findings.

In Chapter 4, we applied the scenario-based approach to the prophet inequality problem. This extension allowed us to model uncertainty in the input distribution as a given set of scenarios, rather than assuming a known distribution. We designed stopping rules with tight guarantees in the worst case, and introduced adaptive, learning-based methods that perform better in practice. Our algorithms were tested on real-world data so as to evaluate their performance in practical settings.

The second part of the thesis, covered in Chapter 5, addressed prediction-based decision-making through the lens of decision theory. We introduced new metrics for evaluating learning-augmented algorithms that go beyond the consistency and the robustness. Our distance-based and risk-aware evaluation criteria allow a more detailed comparison of algorithms under varying levels of prediction accuracy. This framework enables the selection of suitable algorithms from a broader class of candidates, especially when the standard metrics do not distinguish between them. We applied our methodology to well-known online problems such as ski rental, 1-max search, and contract scheduling, and we showed that decision-theoretic tools can guide both the design and the evaluation of learning-augmented algorithms.

Altogether, these contributions demonstrate how robust and learning-augmented approaches can be integrated to address uncertainty in online computation.

Future Work

The results in this thesis open several directions for future research:

Extension to more general data structures. The scenario-based framework could be extended to study tree structures beyond binary search trees and Huffman trees. For example, one could consider B-trees used in databases [BM72] or quad-trees in spatial indexing [FB74], where access costs depend on both structure and dimension. A key difference in B-trees is that internal nodes may have more than two children, and the tree maintains balance by controlling the number of keys per node, which directly impacts access time. Quad-trees, on the other hand, are used to partition two-dimensional space, and are widely applied in spatial data indexing, image processing, and geographic information systems [Sam84a]. In such applications, the distribution of data points may vary depending on context or use-case, such as different user queries or environmental conditions. This makes the scenario-based model particularly relevant, as one may wish to design a single quad-tree structure that performs well across multiple representative spatial distributions. A concrete direction is to analyze the performance of robust B-trees or quad-trees when multiple input scenarios describe different frequency distributions or spatial configurations, and to develop algorithms that minimize the worst-case or average cost across these scenarios.

Combination of predictions with scenario-based uncertainty. A promising direction is to study settings in which predictions depend on the underlying unknown scenario. In such cases, the prediction (e.g., of the maximum value or its position) is scenario-dependent, and the algorithm does not know in advance which scenario applies. This creates the challenge of designing algorithms that are both robust across scenarios and adaptive to the information provided by predictions.

A concrete example is the 1-max search problem [El-96]. The algorithm observes a sequence of values arriving over time and must stop at one of them with the goal of selecting the maximum. Suppose that the values are drawn from one of k distributions, and for each scenario $s \in \{1, \dots, k\}$ a predicted maximum $\hat{p}_s$ is provided. Such predictions could be obtained from machine-learned models trained on historical data under different conditions (e.g., weekday, weekend, or holiday behavior). The algorithm must integrate all predictions to make a stopping decision

that performs well regardless of the true scenario. This setting raises new questions about how to combine multiple forecasts and achieve provable guarantees, such as bounded competitive ratio or low regret, under scenario uncertainty.

Adaptive scenario selection. In some problems, the set of scenarios is not given in advance but can be modeled as something that has to be learned over time. A motivating example is the caching problem with machine-learned predictions [LV21]. The algorithm manages a cache of limited size and must decide which page to evict to minimize misses. For each request, it receives a prediction $\hat{y}(p)$ for the next time page p will be used. These predictions are learned from past data and depend on the current access pattern.

Each access pattern can be modeled as a scenario s_i , with its own request distribution and prediction model. For example, s_1 may correspond to streaming access, while s_2 reflects repeated looping. The true scenario is not known in advance and may change over time. The algorithm starts with a small set of known scenarios and must update or expand this set as it observes new patterns. For example, if the system suddenly sees a burst of repeated requests for a small set of pages, it may add a new scenario s_3 to represent this behavior. This is particularly interesting because many real systems operate in environments that evolve over time, and it is important to design algorithms that can keep up with these changes without losing performance. This raises questions about how to detect and represent new scenarios, when to trust predictions, and how to stay robust while adapting to changes.

Experimental studies on real-world benchmarks. Further experiments on real data would help evaluate how well the proposed methods work in practice. One concrete direction is to apply the robust prophet inequality model (see Chapter 4) to decision-making tasks in finance, such as deciding when to sell a stock or how to divide investments across assets, where the true behavior of returns is uncertain and predictions come from different sources [Kle05; KS77].

Another example comes from sports betting. Suppose we want to place a bet during a football match, and there are three possible outcomes: win, draw, or loss. Each outcome corresponds to a different scenario, and the odds change as the game progresses. The problem is to decide the best time to place the bet without knowing the final result in advance. By observing how the odds evolve in real time, we can test the performance of robust algorithms that aim to make good decisions across all scenarios. This setting offers a natural application of the model and allows for direct comparison with real-world data.

Connections to bandit learning. Another interesting direction is to connect the scenario-based and Bayesian update methods from this thesis with ideas from the multi-armed bandit literature. In a bandit problem, the algorithm repeatedly chooses among several options (arms) and learns from feedback while balancing exploration and exploitation. In our setting, each scenario can be seen as an arm whose true distribution is unknown. As the algorithm observes outcomes, it can update its beliefs about which scenario is more likely, similar to posterior updates in Bayesian bandits. This connection suggests that techniques such as confidence bounds or posterior sampling could be used to design algorithms that explore scenarios while keeping good performance guarantees.

Recent studies also show a growing connection between bandit learning and prophet inequalities. For example, [Gat+24] introduce bandit algorithms for prophet inequality and the Pandora’s Box problem, achieving regret bounds through exploration of uncertain distributions. More recently, [Liu+25] extend these ideas to contextual linear settings, improving regret guarantees and adapting to additional side information. These works open the possibility of viewing the scenario-based prophet model as a repeated decision problem, where the algorithm learns scenario weights over time. Combining bandit-style exploration with our robust and belief-based framework could lead to adaptive algorithms with both theoretical and practical benefits.

Bibliography

- [ADM25] Spyros Angelopoulos, Christoph Dürr, and Georgii Melidi. **Decision-Theoretic Approaches in Learning-Augmented Algorithms**. *CoRR* abs/2501.17701 (2025). DOI: 10.48550/ARXIV.2501.17701. arXiv: 2501.17701. URL: <https://doi.org/10.48550/arXiv.2501.17701> (see page 5).
- [AK22] Makis Arsenis and Robert Kleinberg. **Individual Fairness in Prophet Inequalities**. In: *EC '22: The 23rd ACM Conference on Economics and Computation, Boulder, CO, USA, July 11 - 15, 2022*, 245. DOI: 10.1145/3490486.3538301. URL: <https://doi.org/10.1145/3490486.3538301> (see pages 40, 70).
- [AK23a] Spyros Angelopoulos and Shahin Kamali. **Contract Scheduling with Predictions**. *J. Artif. Intell. Res.* 77 (2023), 395–426. DOI: 10.1613/JAIR.1.14117. URL: <https://doi.org/10.1613/jair.1.14117> (see pages 25, 88, 110, 111, 125).
- [AK23b] Spyros Angelopoulos and Shahin Kamali. **Rényi-Ulam Games and Online Computation with Imperfect Advice**. In: *48th International Symposium on Mathematical Foundations of Computer Science, MFCS*. Vol. 272. LIPIcs, 13:1–13:15 (see page 119).
- [AKW19] Pablo Daniel Azar, Robert Kleinberg, and S. Matthew Weinberg. **Prior independent mechanisms via prophet inequalities with limited information**. *Games Econ. Behav.* 118 (2019), 511–532. DOI: 10.1016/J.GEB.2018.05.006. URL: <https://doi.org/10.1016/j.geb.2018.05.006> (see page 69).
- [AKZ22] Spyros Angelopoulos, Shahin Kamali, and Dehou Zhang. **Online Search with Best-Price and Query-Based Predictions**. In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, February 22 - March 1, 2022*, 9652–9660. DOI: 10.1609/AAAI.V36I9.21199. URL: <https://doi.org/10.1609/aaai.v36i9.21199> (see pages 20, 22, 85, 87, 108, 119).
- [All06] Pieter Allaart. **Prophet Inequalities for I.I.D. Random Variables with Random Arrival Times**. *Sequential Analysis* 26 (Nov. 2006). DOI: 10.1080/07474940701620857 (see page 69).
- [Ana+22] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. **A Regression Approach to Learning-Augmented Online Algorithms**. *CoRR* abs/2205.08717 (2022). DOI: 10.48550/ARXIV.2205.08717. arXiv: 2205.08717. URL: <https://doi.org/10.48550/arXiv.2205.08717> (see page 24).

- [Ang+24a] Spyros Angelopoulos, Marcin Bienkowski, Christoph Dürr, and Bertrand Simon. **Contract Scheduling with Distributional and Multiple Advice**. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, 3652–3660. URL: <https://www.ijcai.org/proceedings/2024/404> (see pages 109, 115).
- [Ang+24b] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. **Online computation with untrusted advice**. *J. Comput. Syst. Sci.* 144 (2024), 103545. DOI: 10.1016/J.JCSS.2024.103545. URL: <https://doi.org/10.1016/j.jcss.2024.103545> (see page 87).
- [Ang+25] Spyros Angelopoulos, Christoph Dürr, Alex Elenter, and Georgii Melidi. **Scenario-Based Robust Optimization of Tree Structures**. In: *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, 26895–26903. DOI: 10.1609/AAAI.V39I25.34894. URL: <https://doi.org/10.1609/aaai.v39i25.34894> (see page 5).
- [Ant+23] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. **Secretary and online matching problems with machine learned advice**. *Discret. Optim.* 48:Part 2 (2023), 100778. DOI: 10.1016/J.DISOPT.2023.100778. URL: <https://doi.org/10.1016/j.disopt.2023.100778> (see pages 17, 20, 85, 128).
- [APT22] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. **Online Graph Algorithms with Predictions**. In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, 35–66. DOI: 10.1137/1.9781611977073.3. URL: <https://doi.org/10.1137/1.9781611977073.3> (see page 17).
- [BD70] Anatole Beck and Newman D. **Yet More on the Linear Search Problem**. *Israel Journal of Mathematics* 8 (Dec. 1970), 419–429. DOI: 10.1007/BF02798690 (see page 24).
- [BD94] Mark S. Boddy and Thomas L. Dean. **Deliberation Scheduling for Problem Solving in Time-Constrained Environments**. *Artif. Intell.* 67:2 (1994), 245–285. DOI: 10.1016/0004-3702(94)90054-X. URL: [https://doi.org/10.1016/0004-3702\(94\)90054-X](https://doi.org/10.1016/0004-3702(94)90054-X) (see page 25).
- [BE98] Allan Borodin and Ran El-Yaniv. **Online computation and competitive analysis**. Cambridge University Press, 1998. ISBN: 978-0-521-56392-5 (see pages 2, 43).
- [Ben+25] Ziyad Benomar, Lorenzo Croissant, Vianney Perchet, and Spyros Angelopoulos. **Pareto-Optimality, Smoothness, and Stochasticity in Learning-Augmented One-Max-Search**. *CoRR* abs/2502.05720 (2025). DOI: 10.48550/ARXIV.2502.05720. arXiv: 2502.05720. URL: <https://doi.org/10.48550/arXiv.2502.05720> (see page 124).

- [BFV22] Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. **A Learned Approach to Design Compressed Rank/Select Data Structures**. *ACM Trans. Algorithms* 18:3 (2022), 24:1–24:28. DOI: 10.1145/3524060. URL: <https://doi.org/10.1145/3524060> (see page 39).
- [BFZ03] Daniel S. Bernstein, Lev Finkelstein, and Shlomo Zilberstein. **Contract Algorithms and Robots on Rays: Unifying Two Scheduling Problems**. In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 1211–1217. URL: <http://ijcai.org/Proceedings/03/Papers/173.pdf> (see page 24).
- [BGN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. **Robust Optimization**. Vol. 28. Princeton Series in Applied Mathematics. Princeton University Press, 2009. ISBN: 978-1-4008-3105-0. DOI: 10.1515/9781400831050. URL: <https://doi.org/10.1515/9781400831050> (see pages 4, 26, 27).
- [BJS09] Niv Buchbinder, Kamal Jain, and Mohit Singh. **Secretary problems and incentives via linear programming**. *SIGecom Exch.* 8:2 (2009), 6. DOI: 10.1145/1980522.1980528. URL: <https://doi.org/10.1145/1980522.1980528> (see page 40).
- [BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. **An Optimal On-Line Algorithm for Metrical Task System**. *J. ACM* 39:4 (1992), 745–763. DOI: 10.1145/146585.146588. URL: <https://doi.org/10.1145/146585.146588> (see page 14).
- [BM72] Rudolf Bayer and Edward M. McCreight. **Organization and Maintenance of Large Ordered Indices**. *Acta Informatica* 1 (1972), 173–189. DOI: 10.1007/BF00288683. URL: <https://doi.org/10.1007/BF00288683> (see page 130).
- [BMS20] Étienne Bamas, Andreas Maggiori, and Ola Svensson. **The Primal-Dual method for Learning Augmented Algorithms**. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL: <https://proceedings.neurips.cc/paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html> (see page 17).
- [Böc+09] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. **On the Advice Complexity of Online Problems**. In: *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*. Vol. 5878. Lecture Notes in Computer Science, 331–340. DOI: 10.1007/978-3-642-10631-6_35. URL: https://doi.org/10.1007/978-3-642-10631-6_35 (see page 3).

- [Boy+17] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. **Online Algorithms with Advice: A Survey**. *ACM Comput. Surv.* 50:2 (2017), 19:1–19:34. DOI: 10.1145/3056461. URL: <https://doi.org/10.1145/3056461> (see page 3).
- [BP25] Ziyad Benomar and Vianney Perchet. **On Tradeoffs in Learning-Augmented Algorithms**. *CoRR* abs/2501.12770 (2025). DOI: 10.48550/ARXIV.2501.12770. arXiv: 2501.12770. URL: <https://doi.org/10.48550/arXiv.2501.12770> (see pages 22, 85, 87, 91, 94, 117).
- [Bru+24a] Johannes Brustle, José Correa, Paul Duetting, and Victor Verdugo. **The Competition Complexity of Dynamic Pricing**. *Math. Oper. Res.* 49:3 (2024), 1986–2008. DOI: 10.1287/MOOR.2022.0230. URL: <https://doi.org/10.1287/moor.2022.0230> (see page 70).
- [Bru+24b] Johannes Brustle, José Correa, Paul Dütting, Tomer Ezra, Michal Feldman, and Victor Verdugo. **The Competition Complexity of Prophet Inequalities**. In: *Proceedings of the 25th ACM Conference on Economics and Computation, EC 2024, New Haven, CT, USA, July 8-11, 2024*, 807–830. DOI: 10.1145/3670865.3673467. URL: <https://doi.org/10.1145/3670865.3673467> (see page 70).
- [BS04] Dimitris Bertsimas and Melvyn Sim. **The Price of Robustness**. *Oper. Res.* 52:1 (2004), 35–53. DOI: 10.1287/OPRE.1030.0065. URL: <https://doi.org/10.1287/opre.1030.0065> (see page 26).
- [BS07] Hans-Georg Beyer and Bernhard Sendhoff. **Robust optimization – A comprehensive survey**. *Computer Methods in Applied Mechanics and Engineering* 196 (July 2007), 3190–3218. DOI: 10.1016/j.cma.2007.03.003 (see page 4).
- [BV04] Stephen Boyd and Lieven Vandenberghe. **Convex optimization**. Cambridge university press, 2004 (see page 55).
- [Cal13] Giuseppe Carlo Calafiore. **Direct data-driven portfolio optimization with guaranteed shortfall probability**. *Autom.* 49:2 (2013), 370–380. DOI: 10.1016/J.AUTOMATICA.2012.11.012. URL: <https://doi.org/10.1016/j.automatica.2012.11.012> (see page 28).
- [CG08] Marco C. Campi and Simone Garatti. **The Exact Feasibility of Randomized Solutions of Uncertain Convex Programs**. *SIAM J. Optim.* 19:3 (2008), 1211–1230. DOI: 10.1137/07069821X. URL: <https://doi.org/10.1137/07069821X> (see page 27).
- [Chr+24] Nicolas Christianson, Bo Sun, Steven H. Low, and Adam Wierman. **Risk-Sensitive Online Algorithms (Extended Abstract)**. In: *The Thirty Seventh Annual Conference on Learning Theory, June 30 - July 3, 2023, Edmonton, Canada*. Vol. 247. Proceedings of Machine Learning Research, 1140–1141.

- URL: <https://proceedings.mlr.press/v247/christianson24a.html> (see pages 88, 91).
- [Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **Introduction to Algorithms, 3rd Edition**. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: <http://mitpress.mit.edu/books/introduction-algorithms> (see page 1).
- [Cor+17] José Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. **Posted Price Mechanisms for a Random Stream of Customers**. In: *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, 169–186. DOI: 10.1145/3033274.3085137. URL: <https://doi.org/10.1145/3033274.3085137> (see page 84).
- [Cor+18] José Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. **Recent developments in prophet inequalities**. *SIGecom Exch.* 17:1 (2018), 61–70. DOI: 10.1145/3331033.3331039. URL: <https://doi.org/10.1145/3331033.3331039> (see page 70).
- [Cor+21] José Correa, Andrés Cristi, Paul Duetting, and Ashkan Norouzi-Fard. **Fairness and Bias in Online Selection**. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Vol. 139. Proceedings of Machine Learning Research, 2112–2121. URL: <http://proceedings.mlr.press/v139/correa21a.html> (see page 70).
- [CSW23] Nicolas Christianson, Junxuan Shen, and Adam Wierman. **Optimal robustness-consistency tradeoffs for learning-augmented metrical task systems**. In: *International Conference on Artificial Intelligence and Statistics, 25-27 April 2023, Palau de Congressos, Valencia, Spain*. Vol. 206. Proceedings of Machine Learning Research, 9377–9399. URL: <https://proceedings.mlr.press/v206/christianson23a.html> (see page 20).
- [Dan+24] Mohammadreza Daneshvaramoli, Helia Karisani, Adam Lechowicz, Bo Sun, Cameron Musco, and Mohammad Hajiesmaili. **Competitive Algorithms for Online Knapsack with Succinct Predictions**. *CoRR* abs/2406.18752 (2024). DOI: 10.48550/ARXIV.2406.18752. arXiv: 2406.18752. URL: <https://doi.org/10.48550/arXiv.2406.18752> (see page 128).
- [Dia+21] Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, Ali Vakilian, and Nikos Zarifis. **Learning Online Algorithms with Distributional Advice**. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Vol. 139. Proceedings of Machine Learning Research, 2687–2696. URL: <http://proceedings.mlr.press/v139/diakonikolas21a.html> (see pages 70, 87, 90, 91, 108).

- [Din+24] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, Aidin Niaparast, and Sergei Vassilvitskii. **Binary Search with Distributional Predictions**. In: *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. URL: http://papers.nips.cc/paper%5C_files/paper/2024/hash/a4b293979b8b521e9222d30c40246911-Abstract-Conference.html (see page 40).
- [DKP09] Stefan Dobrev, Rastislav Královic, and Dana Pardubská. **Measuring the problem-relevant information in input**. *RAIRO Theor. Informatics Appl.* 43:3 (2009), 585–613. DOI: 10.1051/ITA/2009012. URL: <https://doi.org/10.1051/ita/2009012> (see page 3).
- [DL05] Reza Dorriv and Alejandro López-Ortiz. **A Survey of Performance Measures for On-line Algorithms**. *SIGACT News* 36:3 (2005), 67–81. URL: <http://dl.acm.org/citation.cfm?id=1086670> (see page 5).
- [DTA12] Ofer Dekel, Ambuj Tewari, and Raman Arora. **Online Bandit Learning against an Adaptive Adversary: from Regret to Policy Regret**. In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. URL: <http://icml.cc/2012/papers/749.pdf> (see page 88).
- [Düt+20] Paul Dütting, Michal Feldman, Thomas Kesselheim, and Brendan Lucier. **Prophet Inequalities Made Easy: Stochastic Optimization by Pricing Nonstochastic Inputs**. *SIAM J. Comput.* 49:3 (2020), 540–582. DOI: 10.1137/20M1323850. URL: <https://doi.org/10.1137/20M1323850> (see page 69).
- [EFN18] Tomer Ezra, Michal Feldman, and Ilan Nehama. **Prophets and Secretaries with Overbooking**. In: *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, 319–320. DOI: 10.1145/3219166.3219211. URL: <https://doi.org/10.1145/3219166.3219211> (see page 69).
- [EG24] Tomer Ezra and Tamar Garbuz. **The Competition Complexity of Prophet Inequalities with Correlations**. *CoRR* abs/2409.06868 (2024). DOI: 10.48550/ARXIV.2409.06868. arXiv: 2409.06868. URL: <https://doi.org/10.48550/arXiv.2409.06868> (see page 70).
- [El-+01] Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. **Optimal Search and One-Way Trading Online Algorithms**. *Algorithmica* 30:1 (2001), 101–139. DOI: 10.1007/S00453-001-0003-0. URL: <https://doi.org/10.1007/s00453-001-0003-0> (see page 22).

- [El-96] Ran El-Yaniv. **Competitive Solutions for On-line Financial Problems**. In: *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*. Vol. 1442. Lecture Notes in Computer Science, 326–372. DOI: 10.1007/BFB0029576. URL: <https://doi.org/10.1007/BFB0029576> (see pages 21, 130).
- [El-98] Ran El-Yaniv. **Competitive solutions for online financial problems**. *ACM Computing Surveys* 30:1 (Mar. 1998), 28–69. ISSN: 0360-0300. (Visited on 05/21/2024) (see page 99).
- [Ele+24] Alex Elenter, Spyros Angelopoulos, Christoph Dürr, and Yanni Lefki. **Overcoming Brittleness in Pareto-Optimal Learning Augmented Algorithms**. In: *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. URL: http://papers.nips.cc/paper%5C_files/paper/2024/hash/11c6625b0481a7d5625831369f6b7c82-Abstract-Conference.html (see pages 20, 22, 25, 85, 87, 88, 120, 126).
- [Eme+11] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. **Online computation with advice**. *Theor. Comput. Sci.* 412:24 (2011), 2642–2656. DOI: 10.1016/J.TCS.2010.08.007. URL: <https://doi.org/10.1016/j.tcs.2010.08.007> (see page 3).
- [Fab+07] Frank J Fabozzi, Petter N Kolm, Dessislava A Pachamanova, and Sergio M Focardi. **Robust portfolio optimization and management**. John Wiley & Sons, 2007 (see page 26).
- [Faw06] Tom Fawcett. **An introduction to ROC analysis**. *Pattern Recognit. Lett.* 27:8 (2006), 861–874. DOI: 10.1016/J.PATREC.2005.10.010. URL: <https://doi.org/10.1016/j.patrec.2005.10.010> (see page 86).
- [FB74] Raphael A. Finkel and Jon Louis Bentley. **Quad Trees: A Data Structure for Retrieval on Composite Keys**. *Acta Informatica* 4 (1974), 1–9. DOI: 10.1007/BF00288933. URL: <https://doi.org/10.1007/BF00288933> (see page 130).
- [Fis81] Thomas RM Fischer. **On Universal Binary Search Trees**. *Fundamenta Informaticae* 4:1 (1981), 173–184 (see page 40).
- [FSZ24] Chunkai Fu, Jung Hoon Seo, and Samson Zhou. **Learning-Augmented Skip Lists**. *CoRR* abs/2402.10457 (2024). DOI: 10.48550/ARXIV.2402.10457. arXiv: 2402.10457. URL: <https://doi.org/10.48550/arXiv.2402.10457> (see page 39).
- [FV19] Paolo Ferragina and Giorgio Vinciguerra. **Learned Data Structures**. In: *Recent Trends in Learning From Data - Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL 2019)*. Vol. 896. Studies in Computational Intelligence, 5–41. DOI: 10.1007/978-3-030-43883-8_2. URL: https://doi.org/10.1007/978-3-030-43883-8_2 (see page 39).

- [FW98] Amos Fiat and Gerhard J. Woeginger, eds. *Online Algorithms, The State of the Art (the book grew out of a Dagstuhl Seminar, June 1996)*. Vol. 1442. Lecture Notes in Computer Science. Springer, 1998. ISBN: 3-540-64917-4. DOI: 10.1007/BFB0029561. URL: <https://doi.org/10.1007/BFB0029561> (see page 2).
- [Gat+24] Khashayar Gatmiry, Thomas Kesselheim, Sahil Singla, and Yifan Wang. **Bandit Algorithms for Prophet Inequality and Pandora’s Box**. In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, 462–500. DOI: 10.1137/1.9781611977912.18. URL: <https://doi.org/10.1137/1.9781611977912.18> (see page 132).
- [Gel+13] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. **Bayesian Data Analysis**. 3rd. Chapman and Hall/CRC, 2013 (see pages 75, 76).
- [GI03] Donald Goldfarb and Garud Iyengar. **Robust Portfolio Selection Problems**. *Math. Oper. Res.* 28:1 (2003), 1–38. DOI: 10.1287/MOOR.28.1.1.14260. URL: <https://doi.org/10.1287/moor.28.1.1.14260> (see page 4).
- [GJ79] M. R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman, 1979. ISBN: 0-7167-1044-7 (see pages 43, 48, 49).
- [GMS04] Robert Giegerich, Carsten Meyer, and Peter Steffen. **A discipline of dynamic programming over sequence data**. *Sci. Comput. Program.* 51:3 (2004), 215–263. DOI: 10.1016/J.SCICO.2003.12.005. URL: <https://doi.org/10.1016/j.scico.2003.12.005> (see page 58).
- [GMT14] Virginie Gabrel, Cécile Murat, and Aurélie Thiele. **Recent advances in robust optimization: An overview**. *Eur. J. Oper. Res.* 235:3 (2014), 471–483. DOI: 10.1016/J.EJOR.2013.09.036. URL: <https://doi.org/10.1016/j.ejor.2013.09.036> (see page 4).
- [GP19] Sreenivas Gollapudi and Debmalya Panigrahi. **Online Algorithms for Rent-Or-Buy with Expert Advice**. In: *Proceedings of the 36th International Conference on Machine Learning, ICML*. Vol. 97. Proceedings of Machine Learning Research, 2319–2327 (see page 17).
- [GS21] Miri Gilenson and Dvir Shabtay. **Multi-scenario scheduling to maximise the weighted number of just-in-time jobs**. *J. Oper. Res. Soc.* 72:8 (2021), 1762–1779. DOI: 10.1080/01605682.2019.1578628. URL: <https://doi.org/10.1080/01605682.2019.1578628> (see page 27).
- [HK81] Theodore P Hill and Robert P Kertz. **Ratio comparisons of supremum and stop rule expectations**. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 56 (1981), 283–285 (see page 71).

- [HK82] Theodore P Hill and Robert P Kertz. **Comparisons of stop rule and supremum expectations of iid random variables**. *The Annals of Probability* (1982), 336–345 (see page 84).
- [HK92] Theodore P Hill and Robert P Kertz. **A survey of prophet inequalities in optimal stopping theory**. *Contemporary Mathematics* 125:1 (1992), 191 (see page 70).
- [Huf52] David A Huffman. **A method for the construction of minimum-redundancy codes**. *Proceedings of the IRE* 40:9 (1952), 1098–1101. DOI: 10.1109/JRPROC.1952.273898 (see pages 29, 31, 32, 37, 48).
- [HV12] Benjamin Hiller and Tjark Vredeveld. **Probabilistic alternatives for competitive analysis**. *Comput. Sci. Res. Dev.* 27:3 (2012), 189–196. DOI: 10.1007/S00450-011-0149-1. URL: <https://doi.org/10.1007/s00450-011-0149-1> (see page 5).
- [Im+21] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. **Online Knapsack with Frequency Predictions**. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 2733–2743. URL: <https://proceedings.neurips.cc/paper/2021/hash/161c5c5ad51fcc884157890511b3c8b0-Abstract.html> (see page 128).
- [IW91] Makoto Imase and Bernard M. Waxman. **Dynamic Steiner Tree Problem**. *SIAM J. Discret. Math.* 4:3 (1991), 369–384. DOI: 10.1137/0404033. URL: <https://doi.org/10.1137/0404033> (see page 14).
- [JJ10] Li Jialiang and Fine Jason. **Weighted Area Under the Receiver Operating Characteristic Curve and Its Application to Gene Selection**. *Journal of the Royal Statistical Society Series C* 59 (May 2010), 673–692. DOI: 10.1111/j.1467-9876.2010.00713.x (see page 86).
- [Kar+88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. **Competitive Snoopy Caching**. *Algorithmica* 3 (1988), 77–119. DOI: 10.1007/BF01762111. URL: <https://doi.org/10.1007/BF01762111> (see pages 14, 15).
- [Kar+90] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan S. Owicki. **Competitive Randomized Algorithms for Non-Uniform Problems**. In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA*, 301–309. URL: <http://dl.acm.org/citation.cfm?id=320176.320216> (see page 16).
- [Ker86] Robert P Kertz. **Stop rule and supremum expectations of iid random variables: a complete comparison by conjugate duality**. *Journal of multivariate analysis* 19:1 (1986), 88–112 (see page 84).

- [KK18] Jon M. Kleinberg and Robert Kleinberg. **Delegated Search Approximates Efficient Search**. In: *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, 287–302. DOI: 10.1145/3219166.3219205. URL: <https://doi.org/10.1145/3219166.3219205> (see page 69).
- [KKR03] Anna R. Karlin, Claire Kenyon, and Dana Randall. **Dynamic TCP Acknowledgment and Other Stories about $e/(e-1)$** . *Algorithmica* 36:3 (2003), 209–224 (see page 15).
- [KKZ12] Adam Kasperski, Adam Kurpisz, and Pawel Zielinski. **Approximating a two-machine flow shop scheduling under discrete scenario uncertainty**. *Eur. J. Oper. Res.* 217:1 (2012), 36–43. DOI: 10.1016/J.EJOR.2011.08.029. URL: <https://doi.org/10.1016/j.ejor.2011.08.029> (see page 27).
- [KKZ15] Adam Kasperski, Adam Kurpisz, and Pawel Zielinski. **Approximability of the robust representatives selection problem**. *Oper. Res. Lett.* 43:1 (2015), 16–19. DOI: 10.1016/J.ORL.2014.10.007. URL: <https://doi.org/10.1016/j.orl.2014.10.007> (see page 27).
- [Kle05] Robert D. Kleinberg. **A multiple-choice secretary algorithm with applications to online auctions**. In: *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, 630–631. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070519> (see page 131).
- [Knu71] Donald E. Knuth. **Optimum binary search trees**. *Acta informatica* 1 (1971), 14–25. DOI: 10.1007/BF00264289 (see pages 29, 30, 40, 42, 46, 56).
- [Knu97] Donald Ervin Knuth. **The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition**. Addison-Wesley, 1997. ISBN: 0201896834. URL: <https://www.worldcat.org/oclc/312910844> (see page 55).
- [KPS24] Ravi Kumar, Manish Purohit, and Zoya Svitkina. **Improving Online Algorithms via ML Predictions**. *CoRR* abs/2407.17712 (2024). DOI: 10.48550/ARXIV.2407.17712. arXiv: 2407.17712. URL: <https://doi.org/10.48550/arXiv.2407.17712> (see pages 17, 19, 87, 91).
- [KPZ90] Pramod Khargonekar, Ian Petersen, and Keliang Zhou. **Robust stabilization of uncertain linear systems: Quadratic stabilizability and H_∞ control theory**. *IEEE Transactions on Automatic Control - IEEE TRANS AUTOMAT CONTR* 35 (Mar. 1990), 356–361. DOI: 10.1109/9.50357 (see page 26).
- [Kra+18] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. **The Case for Learned Index Structures**. In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, 489–504. DOI: 10.1145/3183713.3196909. URL: <https://doi.org/10.1145/3183713.3196909> (see page 39).

- [KS77] Ulrich Krengel and Louis Sucheston. **Semiamarts and finite values**. *Bulletin of The American Mathematical Society - BULL AMER MATH SOC* 83 (Oct. 1977). DOI: 10.1090/S0002-9904-1977-14378-4 (see pages 23, 131).
- [KS78] Ulrich Krengel and Louis Sucheston. **On semiamarts, amarts, and processes with finite value**. *Probability on Banach spaces* 4:197-266 (1978), 1–2 (see pages 23, 24).
- [KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. **An Optimal Algorithm for On-line Bipartite Matching**. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, 352–358. DOI: 10.1145/100216.100262. URL: <https://doi.org/10.1145/100216.100262> (see page 14).
- [KW12] Robert Kleinberg and S. Matthew Weinberg. **Matroid prophet inequalities**. In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, 123–136. DOI: 10.1145/2213977.2213991. URL: <https://doi.org/10.1145/2213977.2213991> (see page 69).
- [KZ15] Adam Kasperski and Paweł Zieliński. **Complexity of the robust weighted independent set problems on interval graphs**. *Optimization Letters* 9 (2015), 427–436. DOI: 10.1007/s11590-014-0773-3 (see page 27).
- [LAH14] Alejandro López-Ortiz, Spyros Angelopoulos, and Angèle M. Hamel. **Optimal Scheduling of Contract Algorithms for Anytime Problem-Solving**. *J. Artif. Intell. Res.* 51 (2014), 533–554. DOI: 10.1613/JAIR.4360. URL: <https://doi.org/10.1613/jair.4360> (see page 24).
- [Lat+20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. **Online Scheduling via Learned Weights**. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, 1859–1877. DOI: 10.1137/1.9781611975994.114. URL: <https://doi.org/10.1137/1.9781611975994.114> (see page 17).
- [Lec+24] Adam Lechowicz, Rik Sengupta, Bo Sun, Shahin Kamali, and Mohammad Hajiesmaili. **Time Fairness in Online Knapsack Problems**. In: *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. URL: <https://openreview.net/forum?id=9kG7TwgLYu> (see page 40).
- [Lee+24] Russell Lee, Bo Sun, Mohammad Hajiesmaili, and John C. S. Lui. **Online Search with Predictions: Pareto-optimal Algorithm and its Applications in Energy Markets**. In: *The 15th ACM International Conference on Future and Sustainable Energy Systems, e-Energy 2024, Singapore, June 4-7, 2024*. DOI: 10.1145/3632775.3639590. URL: <https://doi.org/10.1145/3632775.3639590> (see page 20).

- [Lin+13] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. **Dynamic Right-Sizing for Power-Proportional Data Centers**. *IEEE/ACM Trans. Netw.* 21:5 (2013), 1378–1391. DOI: 10.1109/TNET.2012.2226216. URL: <https://doi.org/10.1109/TNET.2012.2226216> (see page 14).
- [Liu+25] Junyan Liu, Ziyun Chen, Kun Wang, Haipeng Luo, and Lillian J. Ratliff. **Improved Regret and Contextual Linear Extension for Pandora’s Box and Prophet Inequality**. *CoRR* abs/2505.18828 (2025). DOI: 10.48550/ARXIV.2505.18828. arXiv: 2505.18828. URL: <https://doi.org/10.48550/arXiv.2505.18828> (see page 132).
- [LLW22] Honghao Lin, Tian Luo, and David P. Woodruff. **Learning Augmented Binary Search Trees**. In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Vol. 162. Proceedings of Machine Learning Research, 13431–13440. URL: <https://proceedings.mlr.press/v162/lin22f.html> (see page 39).
- [LM25] Alexander Lindermayr and Nicole Megow. *Repository of works on algorithms with predictions*. <https://algorithms-with-predictions.github.io>. Accessed: 2025-01-01. 2025 (see pages 3, 17).
- [Luc17] Brendan Lucier. **An economic view of prophet inequalities**. *SIGecom Exch.* 16:1 (2017), 24–47. DOI: 10.1145/3144722.3144725. URL: <https://doi.org/10.1145/3144722.3144725> (see page 70).
- [LV21] Thodoris Lykouris and Sergei Vassilvitskii. **Competitive Caching with Machine Learned Advice**. *J. ACM* 68:4 (2021), 24:1–24:25. DOI: 10.1145/3447579. URL: <https://doi.org/10.1145/3447579> (see pages 3, 17, 131).
- [Mar+21] Natalia Martínez, Martín Bertrán, Afroditi Papadaki, Miguel R. D. Rodrigues, and Guillermo Sapiro. **Blind Pareto Fairness and Subgroup Robustness**. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Vol. 139. Proceedings of Machine Learning Research, 7492–7501. URL: <http://proceedings.mlr.press/v139/martinez21a.html> (see page 40).
- [MAS14] Esther Mohr, Iftikhar Ahmad, and Guenter Schmidt. **Online algorithms for conversion problems: A survey**. *Surveys in Operations Research and Management Science* 19 (July 2014), 87–104. DOI: 10.1016/j.sorms.2014.08.001 (see page 22).
- [MBS20] Natalia Martínez, Martín Bertrán, and Guillermo Sapiro. **Minimax Pareto Fairness: A Multi Objective Perspective**. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research, 6755–6764. URL: <http://proceedings.mlr.press/v119/martinez20a.html> (see page 40).

- [Meh75] Kurt Mehlhorn. **Nearly Optimal Binary Search Trees**. *Acta Informatica* 5 (1975), 287–295. DOI: 10.1007/BF00264563. URL: <https://doi.org/10.1007/BF00264563> (see pages 31, 38, 40, 65).
- [Meh84] Kurt Mehlhorn. **Data Structures and Algorithms 1: Sorting and Searching**. Vol. 1. EATCS Monographs on Theoretical Computer Science. Springer, 1984. ISBN: 3-540-13302-X. DOI: 10.1007/978-3-642-69672-5. URL: <https://doi.org/10.1007/978-3-642-69672-5> (see pages 66, 67).
- [Mey05] Adam Meyerson. **The Parking Permit Problem**. In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, 274–284. DOI: 10.1109/SFCS.2005.72. URL: <https://doi.org/10.1109/SFCS.2005.72> (see page 15).
- [MLS22] Dhruv Malik, Yuanzhi Li, and Aarti Singh. **Complete Policy Regret Bounds for Tallying Bandits**. In: *Conference on Learning Theory, 2-5 July 2022, London, UK*. Vol. 178. Proceedings of Machine Learning Research, 5146–5174. URL: <https://proceedings.mlr.press/v178/malik22a.html> (see page 88).
- [MM20] Michal Mankowski and Mikhail Moshkov. **Dynamic programming bi-criteria combinatorial optimization**. *Discret. Appl. Math.* 284 (2020), 513–533. DOI: 10.1016/J.DAM.2020.04.016. URL: <https://doi.org/10.1016/j.dam.2020.04.016> (see page 40).
- [MMS13] Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. **Single machine scheduling with scenarios**. *Theor. Comput. Sci.* 477 (2013), 57–66. DOI: 10.1016/J.TCS.2012.12.006. URL: <https://doi.org/10.1016/j.tcs.2012.12.006> (see page 27).
- [MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. **Competitive Algorithms for On-line Problems**. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, 322–333. DOI: 10.1145/62212.62243. URL: <https://doi.org/10.1145/62212.62243> (see page 14).
- [MR07] Bodo Manthey and Rüdiger Reischuk. **Smoothed analysis of binary search trees**. *Theor. Comput. Sci.* 378:3 (2007), 292–315. DOI: 10.1016/J.TCS.2007.02.035. URL: <https://doi.org/10.1016/j.tcs.2007.02.035> (see page 40).
- [MVT17] Sara Marie McCarthy, Phebe Vayanos, and Milind Tambe. **Staying Ahead of the Game: Adaptive Robust Optimization for Dynamic Allocation of Threat Screening Resources**. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 3770–3776. DOI: 10.24963/IJCAI.2017/527. URL: <https://doi.org/10.24963/ijcai.2017/527> (see page 27).

- [Nan+10] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun (Jim) Xu. **Regret-Minimizing Representative Databases**. *Proc. VLDB Endow.* 3:1 (2010), 1114–1124. DOI: 10.14778/1920841.1920980. URL: <https://vldb.org/pvldb/vol3/R99.pdf> (see page 38).
- [NO13] Ebrahim Nasrabadi and James B. Orlin. **Robust optimization with incremental recourse**. *CoRR* abs/1312.4075 (2013). arXiv: 1312.4075. URL: <http://arxiv.org/abs/1312.4075> (see page 88).
- [PF66] Gilbert John P and Mosteller Frederick. **Recognizing the maximum of a sequence**. *Journal of the American Statistical Association* 61:313 (1966), 35–73 (see page 23).
- [PKL21] Deval Patel, Arindam Khan, and Anand Louis. **Group Fairness for Knapsack Problems**. In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, 1001–1009. DOI: 10.5555/3463952.3464069. URL: <https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1001.pdf> (see page 40).
- [PRC12] Bernardo K. Pagnoncelli, Daniel Reich, and Marco C. Campi. **Risk-Return Trade-off with the Scenario Approach in Practice: A Case Study in Portfolio Selection**. *J. Optim. Theory Appl.* 155:2 (2012), 707–722. DOI: 10.1007/S10957-012-0074-X. URL: <https://doi.org/10.1007/s10957-012-0074-x> (see page 28).
- [PS96] Roberto De Prisco and Alfredo De Santis. **New Lower Bounds on the Cost of Binary Search Trees**. *Theor. Comput. Sci.* 156:1&2 (1996), 315–325. DOI: 10.1016/0304-3975(95)00153-0. URL: [https://doi.org/10.1016/0304-3975\(95\)00153-0](https://doi.org/10.1016/0304-3975(95)00153-0) (see page 65).
- [PSS06] Patrice Perny, Olivier Spanjaard, and Louis-Xavier Storme. **A decision-theoretic approach to robust optimization in multivalued graphs**. *Ann. Oper. Res.* 147:1 (2006), 317–341. DOI: 10.1007/S10479-006-0073-0. URL: <https://doi.org/10.1007/s10479-006-0073-0> (see page 88).
- [RC18] Federico A. Ramponi and Marco C. Campi. **Expected shortfall: Heuristics and certificates**. *Eur. J. Oper. Res.* 267:3 (2018), 1003–1013. DOI: 10.1016/J.EJOR.2017.11.022. URL: <https://doi.org/10.1016/j.ejor.2017.11.022> (see page 28).
- [RS81] Donald B Rosenfield and Roy D Shapiro. **Optimal adaptive price search**. *Journal of Economic Theory* 25:1 (1981), 1–20. ISSN: 0022-0531. DOI: [https://doi.org/10.1016/0022-0531\(81\)90014-4](https://doi.org/10.1016/0022-0531(81)90014-4). URL: <https://www.sciencedirect.com/science/article/pii/0022053181900144> (see page 22).
- [RU+00] R Tyrrell Rockafellar, Stanislav Uryasev, et al. **Optimization of conditional value-at-risk**. *Journal of risk* 2 (2000), 21–42 (see page 90).

- [RU02] R.Tyrrell Rockafellar and Stanislav Uryasev. **Conditional value-at-risk for general loss distributions**. *Journal of Banking and Finance* 26:7 (2002), 1443–1471. ISSN: 0378-4266. DOI: [https://doi.org/10.1016/S0378-4266\(02\)00271-6](https://doi.org/10.1016/S0378-4266(02)00271-6). URL: <https://www.sciencedirect.com/science/article/pii/S0378426602002716> (see page 90).
- [RZ91] Stuart J. Russell and Shlomo Zilberstein. **Composing Real-Time Systems**. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, 212–217. URL: <http://ijcai.org/Proceedings/91-1/Papers/034.pdf> (see pages 24, 25, 109).
- [Sam84a] Hanan Samet. **The Quadtree and Related Hierarchical Data Structures**. *ACM Comput. Surv.* 16:2 (1984), 187–260. DOI: 10.1145/356924.356930. URL: <https://doi.org/10.1145/356924.356930> (see page 130).
- [Sam84b] Ester Samuel-Cahn. **Comparison of threshold stop rules and maximum for independent nonnegative random variables**. *the Annals of Probability* (1984), 1213–1216 (see page 23).
- [SG23] Dvir Shabtay and Miri Gilenson. **A state-of-the-art survey on multi-scenario scheduling**. *Eur. J. Oper. Res.* 310:1 (2023), 3–23. DOI: 10.1016/J.EJOR.2022.11.014. URL: <https://doi.org/10.1016/j.ejor.2022.11.014> (see page 27).
- [Spa03] James C. Spall. **Introduction to stochastic search and optimization - estimation, simulation, and control**. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2003. ISBN: 978-0-471-33052-3. DOI: 10.1002/0471722138. URL: <https://doi.org/10.1002/0471722138> (see page 26).
- [SSU08] Sergey Sarykalin, Gaia Serraino, and Stan Uryasev. “Value-at-risk vs. conditional value-at-risk in risk management and optimization.” In: *State-of-the-art decision-making tools in the information-intensive age*. Informs, 2008, 270–294 (see pages 28, 86).
- [ST19] Yoonchang Sung and Pratap Tokekar. **A Competitive Algorithm for On-line Multi-Robot Exploration of a Translating Plume**. In: *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, 3391–3397. DOI: 10.1109/ICRA.2019.8793850. URL: <https://doi.org/10.1109/ICRA.2019.8793850> (see page 14).
- [ST85a] Daniel D Sleator and Robert E Tarjan. **Amortized efficiency of list update and paging rules**. *Communications of the ACM* 28:2 (1985), 202–208 (see pages 2, 5).
- [ST85b] Daniel Dominic Sleator and Robert Endre Tarjan. **Self-Adjusting Binary Search Trees**. *J. ACM* 32:3 (1985), 652–686. DOI: 10.1145/3828.3835. URL: <https://doi.org/10.1145/3828.3835> (see pages 14, 37).

- [Sun+21] Bo Sun, Russell Lee, Mohammad H. Hajiesmaili, Adam Wierman, and Danny H. K. Tsang. **Pareto-Optimal Learning-Augmented Algorithms for On-line Conversion Problems**. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 10339–10350. URL: <https://proceedings.neurips.cc/paper/2021/hash/55a988dfb00a914717b3000a3374694c-Abstract.html> (see pages 17, 20, 22, 87, 99, 119, 120, 124).
- [Wal45] Abraham Wald. **Statistical Decision Functions Which Minimize the Maximum Risk**. *Annals of Mathematics* 46:2 (1945), 265–280. ISSN: 0003486X, 19398980. URL: <http://www.jstor.org/stable/1969022> (visited on 05/23/2025) (see page 26).
- [Wal50] Abraham Wald. **Statistical Decision Functions**. John Wiley, 1950 (see page 26).
- [Wik24] Wikipedia. *Letter frequency*. https://en.wikipedia.org/wiki/Letter_frequency. [Online; accessed 09-February-2024]. 2024 (see page 60).
- [WZ20] Alexander Wei and Fred Zhang. **Optimal Robustness-Consistency Trade-offs for Learning-Augmented Online Algorithms**. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL: <https://proceedings.neurips.cc/paper/2020/hash/5bd844f11fa520d54fa5edec06ea2507-Abstract.html> (see pages 20, 87, 91).
- [Xin+22] Cao Xinyuan, Chen Jingbang, Chen Li, Lambert Chris, Peng Richard, and Sleator Daniel. **Learning-augmented B-trees**. *arXiv preprint arXiv:2211.09251* (2022) (see page 39).
- [XWL20] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. **An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query**. *VLDB J.* 29:1 (2020), 147–175. DOI: 10.1007/S00778-019-00570-Z. URL: <https://doi.org/10.1007/s00778-019-00570-z> (see page 38).
- [ZKH24] Ali Zeynali, Shahin Kamali, and Mohammad Hajiesmaili. **Robust Learning-Augmented Dictionaries**. In: *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. URL: <https://openreview.net/forum?id=XyhgssAo5b> (see page 39).
- [ZR96] Shlomo Zilberstein and Stuart Russell. **Optimal Composition of Real-Time Systems**. *Artif. Intell.* 82:1-2 (1996), 181–213. DOI: 10.1016/0004-3702(94)00074-3. URL: [https://doi.org/10.1016/0004-3702\(94\)00074-3](https://doi.org/10.1016/0004-3702(94)00074-3) (see page 25).