FAIR ALGORITHMS FOR MULTI-AGENT ENVIRONMENTS

MOHAMMADAMIN SHARIFI

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2026

© MOHAMMADAMIN SHARIFI, 2026

Abstract

In multi-agent environments, k agents often share one solution, each with its own cost function and weight; for example, a binary search tree (BST) with agent-specific key access frequencies. We study algorithmic fairness in such settings: instead of the agents' weighted average cost, we minimize their weighted *maximum* cost, a min-max problem called *Weighted Max Cost Minimization* (WMCM). For shared BSTs, we give deterministic constructions with additive and multiplicative guarantees relative to each agent's optimal tree, prove asymptotically matching lower bounds in specific scenarios, and make them robust to inaccurate access predictions. For general problems, multi-objective optimization results show a deterministic δ-approximation for the weighted-average problem yields only a $k\delta$-approximation for WMCM, a gap inherent to determinism. We show that randomization nearly closes this gap: our framework converts any (δ, β)-approximation algorithm for that problem into a randomized $(\delta, \beta + \epsilon)$-approximation for WMCM, for any $\epsilon > 0$.

Acknowledgements

I would like to thank my supervisor, Professor Shahin Kamali, for his guidance, patience, and support throughout my master's studies. I am also grateful to my committee members, Professor Eric Ruppert (chair) and Professor Ling Jiang, for their time, careful reading of this thesis, and valuable feedback during my defense. This work was supported by a York University graduate scholarship and a research assistantship funded by Professor Shahin Kamali. Finally, I want to thank my family and friends, especially my partner, for their unwavering support and encouragement throughout this journey.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Contributions 4

1.2 Thesis Organization 5

2 Literature Review 6

2.1 Foundations of Algorithmic Fairness 7

2.1.1 Individual versus Group Fairness 7

2.1.2 Fairness Across Application Domains 8

2.2 Algorithms with Predictions 10

2.2.1 The Consistency–Robustness Framework 10

2.2.2 Learning-Augmented Data Structures and Search Trees 11

2.3 Optimal and Shared Binary Search Trees 12

2.3.1 Classical Static Optimality 12

2.3.2 Self-Adjusting Trees . . . 13
2.3.3 Universal Source Coding . . . 13
2.3.4 Worst-Case Fairness and Shared Infrastructure . . . 14
2.4 Multi-Objective Optimization . . . 14
2.4.1 Pareto Optimality . . . 15
2.4.2 The Deterministic Limitation . . . 16
2.5 Minimax Fairness, Game Theory, and the Price of Fairness . . . 16
2.5.1 Maxmin Fairness: Origins . . . 17
2.5.2 Maxmin-Fair Ranking . . . 17
2.5.3 Minimax Duality . . . 18
2.5.4 Fairness-Driven Minimax Problems . . . 19
2.5.5 The Price of Fairness . . . 20
2.6 Summary and Positioning of This Thesis . . . 21

3 Multi-Agent Binary Search Trees 23

3.1 Preliminaries . . . 26
3.2 Additive Optimality . . . 32
3.3 Multiplicative Optimality . . . 35
3.3.1 Layer-by-Layer Construction: Comb . . . 35
3.3.2 Entropy-Weighted Construction: Blend . . . 38
3.3.3 Cross-Entropy-Optimal Construction: Tune . . . 42
3.3.4 Comparison of the Three Algorithms . . . 45
3.4 Robustness to Inaccurate Predictions . . . 46
3.4.1 A General Method: Depth Capping . . . 47
3.4.2 A Sharper Method: Floor and Renormalize . . . 48

4 Multi-Agent Framework 51

4.1 Preliminaries . . . 53

4.2 Main Results 58
4.3 The Robin Algorithm 61
4.3.1 The Algorithm 61
4.3.2 Analysis 65
4.4 The Sherwood Algorithm 72
4.4.1 The Algorithm 72
4.4.2 Analysis 73
4.5 The Marian Algorithm 76
4.5.1 The Algorithm 76
4.5.2 Analysis 78
4.6 Practical Considerations 81

5 Conclusion **83**
5.1 Future Directions 84

Bibliography **85**

Statement on the Use of Generative AI **96**

List of Tables

3.1 Personalized performance guarantees for a shared BST 25

4.1 Running time for the three algorithms. 61

List of Figures

1.1 Three possible BSTs shared between three agents 2

3.1 The three input optimal trees for the COMB example 36

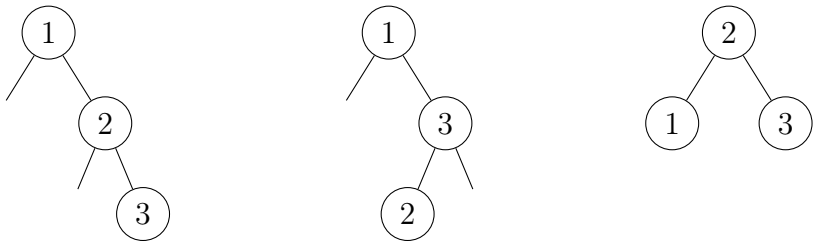
3.2 Result of COMB on the trees of Figure 3.1 37

Chapter 1

Introduction

Optimization problems show up everywhere: any time we want the cheapest or the best solution to a problem, we are solving an optimization problem. Researchers have studied this family of problems for millennia — ancient Greek mathematicians, for instance, studied the isoperimetric problem of finding the maximum area enclosed by a curve of a given length, and the problem of finding the shortest curve between two points in the plane — yet most of that work assumes a single objective to optimize. For example, consider storing a set of keys in a Binary Search Tree (BST), where each key has its own access frequency. Finding the BST that minimizes the average number of comparisons per lookup is a classic optimization problem, solved exactly in polynomial time by Knuth [Knuth, 1971]. In practice, though, systems such as distributed databases and in-memory key-value stores (Redis [Sanfilippo, 2009], DynamoDB [DeCandia et al., 2007], or Cassandra [Lakshman and Malik, 2010]) use dictionaries to store and retrieve data on behalf of several parties at once. The traditional goal of minimizing the *total* cost (latency) summed over all agents does not naturally enforce fairness: total cost can be minimized by a solution that is excellent for one agent and terrible for another one. In many settings, it is desirable that cost be balanced across agents so that no single agent is left far worse off than the others. This raises a natural question: is the resulting solution fair to all of them?

Minimizing the total cost does not, by itself, guarantee fairness. Consider a shared BST with three equally-weighted agents, where agent A_i only accesses the i-th item (Figure 1.1), and A_1 sends far more requests than A_2 or A_3. A system that minimizes total cost places item 1 at the root, giving tree T_1 or T_2 in Figure 1.1. A fair algorithm instead builds a more



(a) T_1 (b) T_2 (c) T_3

Figure 1.1: Three possible BSTs shared between three agents A_1, A_2, and A_3, where A_i only accesses key i. Most accesses come from A_1 (to key 1). T_1 and T_2 are what a system minimizing total cost would build, while T_3 is what a fair system minimizing the maximum cost would build.

balanced tree, such as T_3, where the worst cost across the three agents is as small as possible. T_3 has a higher total cost than T_1 or T_2, but its maximum agent cost is only 2, compared to 3 for T_1 and T_2.

This example raises a natural fairness objective: minimizing the weighted *maximum* cost across agents, rather than their total, an objective we call *Weighted Max Cost Minimization* (WMCM). Note that each agent's weight reflects how much its cost should count toward the fairness objective — for instance, a paying customer with a stronger service-level agreement might receive a higher weight than one on a free tier. For the shared-dictionary example, this means finding a BST that minimizes the worst (weighted) cost paid by any single agent. A good solution under this measure avoids favoring some agents at the expense of others, though this fairness can come at some cost to the total across all agents — a trade-off we quantify later as the *price of fairness* (Section 2.5.5).

We start by studying this fairness question in a concrete and important setting, shared binary search trees, before generalizing it to arbitrary optimization problems. Chapter 3 studies the WMCM problem for shared BSTs in detail: one tree serves k agents, each with its

own access distribution over the keys, and we want this tree to be equally good for all of them at once. We measure how good a tree is for an agent by comparing it to that agent's own best possible tree: the one it would get if it had a tree built just for itself. We compare each agent to its own best possible tree, rather than working with absolute costs directly, because absolute costs are not comparable across agents: two agents with very different access patterns can have very different achievable costs even in the best case, so measuring relative overhead isolates the effect of sharing a single tree from the agents' inherent differences. We use two such comparisons: a *differential* (additive) one, asking how many extra comparisons the agent pays, and a *proportional* (multiplicative) one, asking how many times more expensive the shared tree is.

The key idea behind our constructions is simple: combine the agents' access distributions into a single reference distribution, and build a tree tailored to that combined distribution. We combine the distributions in three ways: a plain average, an entropy-weighted blend, and a combination chosen to minimize the worst-case cross-entropy: each an improvement over the last, at the cost of a more expensive construction. Chapter 3 makes each of these guarantees precise, together with asymptotically matching lower bounds showing that they cannot be improved in specific scenarios.

The "combine and solve" idea behind our BST constructions is not specific to BSTs. Chapter 4 asks the same question for an arbitrary optimization problem shared by k agents, each with its own cost function. There, WMCM (Definition 4.1) is stated formally for this general setting, and paired with its natural counterpart, the *Weighted Average Cost Minimization* (WACM, Definition 4.2) problem: a solution minimizing a weighted *sum* of the agents' costs, exactly the "combine the agents into one and solve" step we used for BSTs. WACM is generally the easier of the two, since it reduces to a single-agent problem with a combined cost function. The key question is how an algorithm that solves, or approximates, WACM can be put to use to solve or approximate WMCM.

Unlike shared BSTs, whose structural flexibility allows us to find a WACM solution that

is simultaneously good for WMCM, this is not always possible in a general deterministic setting. Here is a simple example. Suppose $k-1$ cookies are handed out among k agents, and an agent's cost is 1 if it gets no cookie, and 0 otherwise. Any deterministic algorithm, however clever, leaves at least one agent with no cookie, so it always pays cost 1 under WMCM. A randomized algorithm that hands out the cookies uniformly at random, on the other hand, gives every agent an *expected* cost of only $1/k$. So the best deterministic WMCM cost is 1, while a simple randomized algorithm achieves expected cost $1/k$: a factor-k gap. Every deterministic assignment already achieves the optimal deterministic WACM cost of $1/k$ for this instance (only one agent ever lacks a cookie), so this same example also exhibits a factor-k gap between deterministic WACM and WMCM.

This gap is not just an artifact of the cookie example: in the general setting, the best known bound is due to Glaßer et al. [Glaßer et al., 2010]: any *deterministic* algorithm giving a δ-approximation for WACM guarantees a deterministic $k\delta$-approximation for WMCM, where k is the number of agents. This factor of k reflects a genuine limitation of deterministic algorithms in this setting, not merely a loose analysis. This kind of gap between deterministic and randomized fairness shows up often when designing fair algorithms (see, e.g., [Bercea et al., 2019, Correa et al., 2021]).

This is why we turn to randomization in Chapter 4: using randomization, we show the factor-k gap can be closed almost entirely, turning any approximation algorithm for WACM into basically the same approximation for WMCM, with only a small, controllable additive term.

1.1 Contributions

Our main contributions are as follows.

Fair Binary Search Trees. We study the shared-BST problem directly (Chapter 3), giving deterministic constructions with both additive and multiplicative guarantees relative

to each agent's own optimal tree. Our three multiplicative constructions (COMB, BLEND, and TUNE) trade off simplicity for tightness, each improving on the last at the cost of a more expensive construction, and we prove asymptotically matching lower bounds showing that, in specific scenarios, each guarantee cannot be improved further. We also show how to make every construction robust to inaccurate predictions about the agents' access patterns.

A General Randomized Framework. We give a framework (Chapter 4) that takes any (δ, β)-approximation algorithm for WACM (Weighted Average Cost Minimization), meaning one whose cost is within a multiplicative factor δ and an additive term β of the best possible, and turns it into a randomized $(\delta, \beta + \epsilon)$-approximation algorithm for WMCM (Weighted Max Cost Minimization), for any desired precision $\epsilon > 0$.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 reviews related work on fairness in algorithm design, multi-objective optimization, and related topics. Chapter 3 studies the deterministic multi-agent binary search tree problem in detail. Chapter 4 then presents the general multi-agent framework: the randomized algorithms themselves, their analysis, and full proofs of the approximation guarantees.

Chapter 2

Literature Review

This chapter surveys the literature underlying the rest of the thesis. It is organized around the two concepts studied later: shared binary search trees in Chapter 3, and the general multi-agent fairness framework of Chapter 4 that generalizes it. Section 2.1 lays out the general vocabulary of algorithmic fairness that the rest of the chapter refines. Section 2.2 covers algorithms with predictions, including the learning-augmented search-tree literature most relevant to Chapter 3. Section 2.3 reviews the classical and modern binary-search-tree literature that Chapter 3 builds on directly. Section 2.4 covers multi-objective optimization and the deterministic sum-versus-max gap that motivates our randomized framework. Section 2.5 traces the maxmin fairness objective from its roots in economics and game theory through the online-learning and convex-optimization tools we use to compute it, and the price it pays for fairness, including a discussion of maxmin-fair ranking, the closest prior work to our own (Section 2.5.2). Section 2.6 positions the thesis relative to all of the above.

2.1 Foundations of Algorithmic Fairness

2.1.1 Individual versus Group Fairness

There are two main ways researchers think about fairness for algorithms. *Individual fairness*, introduced by Dwork et al. [Dwork et al., 2012], says that similar individuals should be treated similarly, where similarity is measured by some task-specific metric. This is a natural idea, but it depends on having a good similarity metric, and in practice one is often unavailable. *Group fairness*, used for example in the equalized-odds and equal-opportunity criteria of Hardt et al. [Hardt et al., 2016], instead compares aggregate statistics, such as a classifier's error rate, across demographic groups. This avoids the need for a similarity metric, but it comes at a cost: it says nothing about how any single individual is treated.

Given this, we need a fairness notion that fits our setting. Chapter 3 makes this precise using either the ratio or the difference between an agent's cost under the shared tree and its own optimal cost. Chapter 4 then generalizes this to a *worst-agent* (maxmin) criterion over arbitrary per-agent cost functions: minimizing the maximum cost across agents with their own specific cost functions. Both are closer to group fairness in spirit, since they compare agents to each other rather than to an outside metric: an agent in our setting can be anything from a single individual to an entire group of people, so when an agent represents a group, comparing agents' costs amounts to comparing an aggregate statistic across groups, exactly as group fairness does. But because they are checked for every agent rather than only on average, they keep some of the individual-level protection that group fairness lacks.

Another notion of fairness, mostly relevant to online decision-making, is *time fairness*, which requires that otherwise comparable requests not be advantaged or disadvantaged solely because of their arrival time; variants of this notion have been studied in secretary and online-selection problems [Arsenis and Kleinberg, 2022], as well as in the online knapsack problem [Lechowicz et al., 2024].

2.1.2 Fairness Across Application Domains

Fairness constraints of the group-fairness flavor have been layered onto a wide range of classical optimization problems. We survey clustering, matching, scheduling, and selection problems, since our multi-agent framework (Chapter 4) applies to all of them.

Fair Clustering. In fair clustering, data points carry group labels, and the goal is to minimize clustering cost while maintaining balanced group representation within each cluster. Chierichetti et al. [Chierichetti et al., 2017] introduced *fairlets* (small, balanced building-block clusters) and showed that any clustering algorithm can be made fair by first decomposing the data into fairlets, bounding the resulting price of fairness for two groups. Bera et al. [Bera et al., 2019] extended this to an arbitrary number of protected groups and to k-center, k-median, and k-means simultaneously, via a reduction to matroid intersection; Kleindessner et al. [Kleindessner et al., 2019a] specialized the fair k-center problem for data summarization. Rösner and Schmidt [Rösner and Schmidt, 2018] connected fair clustering to privacy-preserving analysis, and Ahmadian et al. [Ahmadian et al., 2020] gave a constant-factor approximation for fair correlation clustering. Kleindessner et al. [Kleindessner et al., 2019b] extended fairness guarantees to spectral clustering. Bercea et al. [Bercea et al., 2019] and Esmaeili et al. [Esmaeili et al., 2021] quantified the price of fairness in clustering directly, showing that the cost increase from fairness constraints is often modest in practice even though it can be unbounded in the worst case.

A distinct and, for us, more relevant strand is *socially fair* clustering, where the objective is to minimize the *maximum* cost across demographic groups rather than the total cost, which is precisely our WMCM objective specialized to clustering. Makarychev and Vakilian [Makarychev and Vakilian, 2021] gave an $O(\log \ell / \log \log \ell)$-approximation for socially fair k-median and k-means, where ℓ is the number of groups; Ghadiri et al. [Ghadiri et al., 2021] improved this to a constant-factor approximation for socially fair k-means; Goyal and Jaiswal [Goyal and Jaiswal, 2023] gave a tight FPT approximation; and Micha and Shah [Micha and Shah, 2020]

revisited proportionally fair clustering under a different, share-based notion of fairness. This line of work is the clustering-specific counterpart of the general question we study: how much does a shared solution have to sacrifice to avoid favoring some group (or agent) over another?

Fair Matching. García-Soriano and Bonchi [García-Soriano and Bonchi, 2020] studied fair-by-design bipartite matching, where fairness is individual rather than group-based: since many maximum matchings may exist, each satisfying a different set of individuals, they define a matching distribution to be maxmin-fair if no individual's matching probability can be improved without decreasing that of someone no better off, showing such distributions can be computed efficiently via convex optimization. Theirs is a precursor to the maxmin-fair ranking framework of the same authors, discussed at length in Section 2.5.2. Sankar et al. [Sankar et al., 2021] studied group-fairness constraints in online and offline matching, and Hosseini et al. [Hosseini et al., 2023] studied class-based fairness constraints in online bipartite matching.

A related, deterministic notion is *envy-freeness*: a matching is envy-free if no unmatched (or worse-off) agent on one side would prefer another agent's match to their own. Wu and Roth [Wu and Roth, 2018] studied the lattice structure of envy-free matchings in the many-to-one (hospitals-doctors) setting, and Aigner-Horev and Segal-Halevi [Aigner-Horev and Segal-Halevi, 2022] gave a polynomial-time algorithm for maximum-cardinality envy-free matchings in bipartite graphs and connected the notion to fair division.

Fair Scheduling. Minimizing the maximum machine load, the classical *makespan*, is itself a minimax objective, and its study predates the fairness literature by decades: Graham [Graham, 1969] gave the first worst-case bounds for list scheduling, and Lenstra, Shmoys, and Tardos [Lenstra et al., 1990] gave a 2-approximation for makespan minimization on unrelated parallel machines together with a matching hardness result. Agnetis et al. [Agnetis et al., 2004] studied scheduling with *two competing agents* sharing a single machine, each with its own objective: a two-agent special case of the shared-resource setting this thesis generalizes

to k agents. More recently, Im and Moseley [Im and Moseley, 2017] gave a constant-factor approximation for minimizing flow-time norms via a fairness-motivated sampling scheme.

Fair Selection: Knapsack and the Secretary Problem. In the proportional online knapsack problem, items of size at most ϵ and arbitrary value arrive online, and a natural *threshold algorithm* accepts an item once its value-to-size ratio clears an increasing threshold. Because the threshold rises over time, two identical items can be treated differently depending on when they arrive. This is a violation of what Lechowicz et al. [Lechowicz et al., 2024] call *time fairness*: treating equally qualified items consistently regardless of arrival order. They showed predictions are necessary and sufficient to restore this guarantee. Bercea et al. [Bercea et al., 2019] and Correa et al. [Correa et al., 2021] document similar phenomena in cost-based and sampling-based selection settings.

2.2 Algorithms with Predictions

Algorithms with predictions, also called *learning-augmented algorithms*, give an algorithm access to a machine-learned prediction about its input, and ask for improved performance when the prediction is accurate without sacrificing a worst-case guarantee when it is not. This framework is directly relevant to Chapter 3, whose shared-BST algorithms are themselves constructed using predictions of agent access patterns, made robust in Section 3.4 against exactly this kind of inaccuracy.

2.2.1 The Consistency–Robustness Framework

Lykouris and Vassilvitskii [Lykouris and Vassilvitskii, 2021] crystallized the now-standard framework for this area in the context of competitive caching: an algorithm should be *consistent* (near-optimal when predictions are accurate) and *robust* (bounded competitive ratio even when predictions are arbitrarily wrong), with performance degrading gracefully as

prediction error grows. Purohit et al. [Purohit et al., 2018] applied the same framework to ski rental and non-clairvoyant scheduling, parameterizing performance directly by prediction error; Wei and Zhang [Wei and Zhang, 2020] characterized the optimal achievable robustness–consistency trade-off in general; and Bamas et al. [Bamas et al., 2020] gave a primal-dual method for designing learning-augmented algorithms systematically rather than case by case. Mitzenmacher and Vassilvitskii [Mitzenmacher and Vassilvitskii, 2020] survey the resulting body of work across caching, scheduling, online optimization, graph problems, and data structures.

2.2.2 Learning-Augmented Data Structures and Search Trees

A second line of work uses predictions specifically to improve indexing and search, and is the most directly relevant prior work for the shared-BST problem of Chapter 3. Kraska et al. [Kraska et al., 2018] introduced *learned indexes*, replacing classical index structures with models trained to predict a key's position, and Ferragina et al. [Ferragina et al., 2020] gave a theoretical account of why learned indexes are effective. Closer to our setting, Lin, Luo, and Woodruff [Lin et al., 2022] and Chen et al. [Chen et al., 2025] studied *learning-augmented binary search trees*, in which a single predicted access distribution is used to build a tree that is consistent with an accurate prediction and robust to an inaccurate one. Related predictions have been incorporated into list labeling [McCauley et al., 2023], priority queues [Benomar and Coester, 2024], caching [Lykouris and Vassilvitskii, 2018], and dictionaries [Zeynali et al., 2024]; the last of these, Zeynali, Kamali, and Hajiesmaili's *robust learning-augmented dictionaries*, targets a closely related skip-list-based structure.

In every one of these works, however, the prediction is a *single* global workload, evaluated against a *single* offline optimum. The shared-BST problem of Chapter 3 instead starts from k separate predicted access distributions, one per agent, and asks for a single tree that is simultaneously good against k separate personalized optima. This is a genuinely harder problem even when every distribution is known exactly, since (as Chapter 3 demonstrates)

no single tree can be optimal for two agents whose distributions disagree. Section 3.4 does provide robustness guarantees in the classical single-prediction sense, bounding the worst-case depth of every item regardless of prediction accuracy.

2.3 Optimal and Shared Binary Search Trees

This section reviews the classical and modern binary-search-tree literature underlying Chapter 3, our specialization of the multi-agent framework to shared BSTs. Four bodies of work are relevant: classical static optimality, self-adjusting trees, universal source coding (whose objective turns out to have exactly the same shape as our TUNE algorithm), and worst-case notions of fairness for shared infrastructure.

2.3.1 Classical Static Optimality

For a single access distribution, the optimal BST problem is classical. Knuth [Knuth, 1971] gave an $O(n^2)$-time dynamic program computing the exact optimum, exploiting a monotonicity property of optimal roots that reduces the naive $O(n^3)$ recurrence. Mehlhorn [Mehlhorn, 1975, 1984] gave an $O(n \log n)$-time construction that places each item x within an additive constant of its own information-theoretic ideal depth $-\log p(x)$, where $p(x)$ denotes the access frequency of x, together with matching entropy lower bounds relating the optimal cost to the Shannon entropy of the access distribution; Bayer [Bayer, 1975] sharpened these lower bounds further. Chapter 3 uses these results as blackbox tools, most directly Mehlhorn's construction (Lemma 3.7) and the resulting entropy characterization (Lemma 3.6), rather than as competing solutions to our problem: each assumes a single distribution and produces a tree tailored to it, whereas we are given k distributions that one tree must serve simultaneously.

2.3.2 Self-Adjusting Trees

Sleator and Tarjan's splay trees [Sleator and Tarjan, 1985] take a different approach entirely: rather than fixing a tree to a predicted distribution, the tree restructures itself after every access, and is analyzed against an *offline* benchmark for a single observed request sequence rather than a fixed distribution. This makes splay trees a natural experimental baseline, but not a competitor in our theoretical setting: a splay tree can serve a mixed multi-agent request sequence well in aggregate while treating individual agents within that sequence very unevenly, since its guarantee says nothing about per-agent cost.

2.3.3 Universal Source Coding

Universal source coding asks a structurally similar question in a different guise: several distributions are candidates to be the "active" one, a single code must be committed to in advance, and the goal is to minimize the worst-case excess length relative to the code that could have been built knowing the active distribution. If a code is built from distribution q and distribution p_i turns out to be active, the excess length is exactly the Kullback–Leibler divergence $D_{\mathrm{KL}}(p_i \| q)$, a standard information-theoretic measure of how much one distribution differs from another [Cover and Thomas, 2006], so the best worst-case code minimizes $\max_i D_{\mathrm{KL}}(p_i \| q)$ over q. The classical universal-coding literature shows that the minimizing q is always a mixture of the candidate distributions [Davisson, 1973, Gallager, 1976, Merhav and Feder, 1995, 1998], computable via the Blahut–Arimoto iteration [Blahut, 1972, Arimoto, 1972], originally developed to compute exactly this minimax value.

A similar minimax objective appears in our TUNE algorithm (Section 3.3.3) to compute a candidate reference distribution q; we solve this optimization problem using the existing methods described above.

2.3.4 Worst-Case Fairness and Shared Infrastructure

Our worst-agent objective is also connected to two lines of prior work outside the data structures domain. In machine learning, minimax group fairness minimizes the largest loss over groups rather than the average, studied by Martinez et al. [Martinez et al., 2020] and Diana et al. [Diana et al., 2021]; both compare a group's loss to the losses of other groups under the same model, rather than to a personalized counterfactual model built for that group alone. In distributed systems, sharing a resource among heterogeneous tenants is a long-standing concern: Ghodsi et al. [Ghodsi et al., 2011] introduced Dominant Resource Fairness for allocating multiple resource types (CPU, memory, etc.) across tenants with different demand profiles; Kunjir et al. [Kunjir et al., 2017] introduced ROBUS, a fair cache-allocation policy for data-parallel workloads; and Aulbach et al. [Aulbach et al., 2008] studied schema-mapping techniques for multi-tenant databases, the systems context (shared indexes and storage) closest to our shared-BST setting.

All three compare agents either to one another or to an external axiom such as envy-freeness. We instead compare each agent to a *personalized counterfactual*: the cost it would have achieved from a structure built exclusively for it. This makes the guarantee independent of how the other agents happen to be constituted, since an agent's guarantee does not degrade merely because another agent's workload changes. It also makes the comparison harder to establish, because the counterfactual C_i is itself the solution of an optimization problem rather than an observable quantity.

2.4 Multi-Objective Optimization

Multi-objective optimization (MOO) studies problems with several simultaneous, generally conflicting objectives, including multi-objective shortest path [Warburton, 1987, Raith and Ehrgott, 2009] and minimum spanning tree [Hamacher and Ruhe, 1994]. Unlike single-objective optimization, MOO problems typically admit no single optimal solution; the goal

instead is to characterize the trade-off between objectives. This section reviews that theory and its central hardness result, the factor-k gap between minimizing the sum of costs and minimizing the maximum cost, which is the starting point for the framework of Chapter 4.

2.4.1 Pareto Optimality

A solution s is *Pareto optimal* if no feasible s' improves one cost function without worsening another: formally, given cost functions $C_1, \ldots, C_k$ over a feasible set $\mathcal{S}$, no $s' \in \mathcal{S}$ satisfies $C_i(s') \leq C_i(s)$ for all i and $C_j(s') < C_j(s)$ for some j. The *Pareto set* of all such solutions is the natural object of study in MOO, but it can be exponentially large. The number of Pareto-optimal paths in the biobjective shortest path problem, for instance, can be exponential in the number of vertices [Ehrgott, 2005], so exact computation is generally intractable.

Papadimitriou and Yannakakis [Papadimitriou and Yannakakis, 2000] showed that this intractability can be circumvented approximately: for any $\epsilon > 0$, there is a $(1+\epsilon)$-approximate Pareto set of size polynomial in the input and $1/\epsilon$, meaning every Pareto-optimal solution is dominated within a $(1+\epsilon)$ factor, on every objective, by some solution in the approximate set. They further showed this set is computable in polynomial time whenever an auxiliary decision problem, GAP, is polynomial-time solvable: given a target cost vector $\mathbf{b}$ and tolerance δ, GAP asks for a solution with cost at most b_i on every objective i, or a certificate that no solution achieves cost b_i/δ on every objective. Diakonikolas and Yannakakis [Diakonikolas and Yannakakis, 2009] later sharpened the size bounds for approximate Pareto sets specifically in the biobjective shortest-path setting, and Vassilvitskii and Yannakakis [Vassilvitskii and Yannakakis, 2004] gave algorithms for computing succinct trade-off curves directly.

GAP turns out to be exactly the deterministic WMCM problem in disguise: a δ-approximation for one yields a δ-approximation for the other [Glaßer et al., 2010]. This equivalence is the technical bridge between the Pareto-approximation literature and the min-max fairness objective this thesis studies, and it is what lets the hardness result discussed next transfer directly to WMCM.

2.4.2 The Deterministic Limitation

Glaßer et al. [Glaßer et al., 2010] gave the most general deterministic reduction between WACM and WMCM: any deterministic δ-approximation for WACM (equivalently GAP) yields a deterministic $k\delta$-approximation for WMCM, where k is the number of objectives (agents). This factor of k is not merely an artifact of their reduction: the cookie example of Chapter 1, where $k-1$ indivisible items are split among k agents, already shows that no deterministic algorithm can do better in general, since every deterministic assignment achieves the optimal WACM cost of $1/k$ while still paying WMCM cost 1 — exactly the factor-k gap.

This tightness result is the central motivation for the shift to randomization in Chapter 4. It establishes that any deterministic algorithm for shared, multi-agent optimization must pay a cost that degrades linearly in the number of agents, no matter how good the underlying single-objective solver is. The natural question is whether this factor-k loss is inherent to *all* algorithms, or only to deterministic ones. As we show in Chapter 4, it is the latter: our randomized framework turns any approximation algorithm for WACM into an almost identical approximation for WMCM, with a controllable additive difference $\epsilon > 0$, eliminating the multiplicative-k loss entirely.

2.5 Minimax Fairness, Game Theory, and the Price of Fairness

Minimizing the maximum cost, or equivalently maximizing the minimum utility, is one of the oldest and most widely used fairness criteria, with roots that predate algorithmic fairness by decades. This section traces those roots, the game-theoretic and online-learning tools used to compute maxmin-fair solutions (which Chapter 4 draws on directly), and the price such solutions pay relative to unconstrained optimization.

2.5.1 Maxmin Fairness: Origins

Rawls [Rawls, 1971] articulated the *difference principle*: social and economic arrangements should be judged by their effect on the least-advantaged member of society. Applied to algorithm design, this becomes the maxmin objective at the center of this thesis: maximize the minimum utility (minimize the maximum cost) across all agents. In resource allocation, Kelly, Maulloo, and Tan [Kelly et al., 1998] introduced the related but distinct notion of *proportional fairness*, under which no agent can increase its rate without a proportionally larger aggregate decrease elsewhere, closely tied to the Nash bargaining solution [Nash, 1950] and widely adopted in network engineering; Banerjee et al. [Banerjee et al., 2023] extended proportional fairness to the online allocation of public goods. Nguyen et al. [Nguyen et al., 2024] generalized proportional fairness beyond network allocation to general combinatorial optimization, a generalization in the same spirit as, though technically disjoint from, the maxmin generalization this thesis pursues.

The canonical hard instance for maxmin fairness in scheduling is the *Santa Claus problem* of Bansal and Sviridenko [Bansal and Sviridenko, 2006]: divide indivisible presents among m children to maximize the happiness of the least happy child, an NP-hard problem for which they gave an $O(\log\log m/\log\log\log m)$-approximation via a clever LP relaxation and rounding. Gupta, Moondra, and Singh [Gupta et al., 2023] studied the analogous max-min fairness objective in facility location. Both are, in our terminology, instances of WMCM for specific problems, and both illustrate that even when the single-agent (or single-objective) problem is easy, its maxmin multi-agent generalization can become substantially harder, the same phenomenon we formalize for shared binary search trees in Chapter 3.

2.5.2 Maxmin-Fair Ranking

The closest prior work to our framework is García-Soriano and Bonchi's maxmin-fair ranking framework [García-Soriano and Bonchi, 2021]. A *ranking* here is an ordering of a set of individuals, for example candidates for a set of positions, where each ordering incurs its

own cost to each individual, and they studied rankings that must also satisfy group-fairness constraints. Their question was how much such a constrained ranking can cost a single individual, and whether randomizing over rankings can lower that cost. They answered yes: using the ellipsoid method, they computed, in polynomial time, a randomized ranking that minimizes the cost to the worst-off individual.

2.5.3 Minimax Duality

Von Neumann's minimax theorem [von Neumann and Morgenstern, 1944] establishes that in a zero-sum two-player game with payoff matrix A,

$$\max_{\mathbf{p}} \min_{\mathbf{q}} \mathbf{p}^\top A \mathbf{q} = \min_{\mathbf{q}} \max_{\mathbf{p}} \mathbf{p}^\top A \mathbf{q},$$

so that the order in which the two players commit to a (possibly randomized) strategy does not affect the game's value. Wald's generalization [Wald, 1945] extends this equality beyond finite matrix games to the more general minimax problems this thesis relies on: it is what allows a minimax problem to be solved by repeatedly solving a sequence of simpler, ordinary optimization problems, rather than tackling the minimax objective directly.

One way to compute this shared value algorithmically is *Hedge*, the classical multiplicative-weights algorithm from online learning, due to Freund and Schapire [Freund and Schapire, 1999] and surveyed as a general algorithmic meta-tool by Arora, Hazan, and Kale [Arora et al., 2012]. In our setting, the two players of the zero-sum game are an *optimizer*, whose strategies are candidate solutions, and an *adversary*, whose strategies are the k agents; the payoff matrix entry for a given solution and agent is that agent's cost under the solution, and a mixed strategy for the adversary is exactly a weighting of the agents (Chapter 4 makes this precise). In our multi-agent fairness setting, Hedge maintains a weight on each agent and, after every round, increases the weight of agents that incurred a larger loss, so that future rounds favor whichever agent has fared worst so far. When both sides of the zero-sum game

are updated this way, the resulting sequence of play converges, on average, to exactly the value the minimax theorem above guarantees exists. Syrgkanis et al. [Syrgkanis et al., 2015] showed that when both sides use such a no-regret update rule, this convergence accelerates: the gap to the shared value after T rounds shrinks from $O(1/\sqrt{T})$ to $O(1/T)$.

A different route to the same minimax value is the *ellipsoid method*. Starting from a region guaranteed to contain the optimal solution, the method repeatedly checks whether the region's center is feasible and, if not, uses a violated constraint to cut away roughly half the region, shrinking the search space iteratively until only a small region containing the optimum remains. This classical equivalence between optimization and separation is due to Khachiyan and to Grötschel, Lovász, and Schrijver [Khachiyan, 1979, Grötschel et al., 1988]. The ellipsoid method is not the only algorithm built on this shrinking-search-space idea: Vaidya [Vaidya, 1996] gave an interior-point-style algorithm with better dependence on the dimension, and Kelley's classical cutting-plane method [Kelley, 1960] is the primal-side counterpart of the ellipsoid's dual cuts.

A related oracle-based approach outside fairness is due to Hellerstein, Lidbetter, and Pirutinsky [Hellerstein et al., 2017], who study minimax search games where a maximizing player with n pure strategies faces a minimizing player accessed only through an α-approximate best-response oracle, and give both an ellipsoid-based and a multiplicative-weights algorithm for computing near-optimal strategies.

2.5.4 Fairness-Driven Minimax Problems

Several works study minimax objectives motivated explicitly by fairness. The *k-server problem* is a classical online problem in which k mobile servers, placed in a metric space, must be moved to serve a sequence of requests arriving one at a time, at a cost equal to the distance traveled [Manasse et al., 1990]; a recent line of work studies how to make this movement cost fair across servers [Daneshvaramoli et al., 2025]. Singh et al. [Singh et al., 2024] introduced the Fair k-Food problem, a fairness-driven variant of the k-server problem in which the

goal is to maximize the minimum reward earned by any delivery agent in a food-delivery network. Goel and Meyerson [Goel and Meyerson, 2006] showed that for problems with concave profits or convex costs, all objectives can be simultaneously approximated within a constant factor, a general condition under which minimax approximation is tractable. The socially fair clustering literature discussed in Section 2.1.2 is another instance of this pattern: minimizing the maximum clustering cost across demographic groups is precisely a WMCM instance.

2.5.5 The Price of Fairness

The *price of fairness* measures the efficiency lost by imposing a fairness constraint. Bertsimas et al. [Bertsimas et al., 2011] gave a systematic treatment for resource allocation, showing the price depends on both the fairness notion chosen (proportional or maxmin) and the structure of the feasible set. Caragiannis et al. [Caragiannis et al., 2012] bounded the price of fairness for several fair-division objectives, and Bercea et al. [Bercea et al., 2019] and Esmaeili et al. [Esmaeili et al., 2021] did so specifically for clustering (Section 2.1.2).

In Chapter 3, we show the price of fairness for our multiplicative objective is $\log k$ in the deterministic setting: there is always a shared binary search tree whose cost for every agent is within a $\log k$ factor of that agent's own optimal cost, and this factor is tight in the worst case, even against randomized algorithms.

Chapter 4 gives a different answer to a related question for our general framework, allowing randomization. We show there is always a weighting of the agents for which the optimal WACM cost equals the optimal WMCM cost, and our algorithms find such a weighting up to an arbitrarily small additive slack ϵ. Measured against this best achievable WACM solution rather than the unconstrained optimum, the price of fairness is therefore essentially just ϵ: near zero and independent of the number of agents k, in contrast to the deterministic setting, where it can grow linearly with k. This does not conflict with the $\log k$ bound above: that bound is the best possible deterministic guarantee for shared BSTs, while the near-zero price

of fairness here is achieved only by allowing randomization.

2.6 Summary and Positioning of This Thesis

Four themes recur across the literature surveyed in this chapter, and each shapes a specific technical choice made later in the thesis: fairness, algorithms with predictions, binary search trees, and multi-objective (minmax) optimization.

Fairness. Fairness is ubiquitous but domain-specific: clustering, matching, scheduling, ranking, and data structures each have their own fair variant, typically developed independently (Sections 2.1 and 2.3). Our framework (Chapter 4) is deliberately domain-agnostic in response, applying to any problem with a weighted-sum solver, and it does so by leaning on a pattern that already recurs in the literature: reducing a hard fairness objective to a sequence of easier weighted-sum problems, as in García-Soriano and Bonchi's maxmin-fair ranking (Section 2.5.2). Our treatment differs from theirs mainly in scope, applying to any problem with a weighted-sum solver rather than just ranking, tolerating an *approximate* oracle rather than requiring an exact one, and offering three algorithms (Chapter 4) with different trade-offs between the number of agents and the desired precision.

Algorithms with predictions. A separate line of work asks how to exploit a machine-learned prediction of the input while still guaranteeing good worst-case behavior when that prediction is wrong (Section 2.2). Chapter 3 builds on this directly: each of our BST constructions comes with a robust variant (Section 3.4) that retains its personalized guarantee when the supplied access distribution is accurate, while still bounding worst-case depth when it is not.

Binary search trees. The classical BST literature (Section 2.3.1) and the learning-augmented BST literature discussed above (Section 2.2.2) share a common assumption: both

optimize a tree against a single, known or predicted access distribution. Chapter 3 shows that serving k such distributions *simultaneously*, against k separate personalized objectives, is a genuinely different and generally harder problem, one for which, to our knowledge, no prior lower bounds were known.

Multi-objective and minmax optimization. The central tension this thesis responds to is already visible here: the factor-k gap between deterministic WACM and WMCM approximations (Section 2.4.2) cannot be closed by any deterministic algorithm. Chapter 4 shows the gap *can* be closed with randomization, at the cost of only a controllable additive term, and Chapter 3 shows that studying a specific problem directly, rather than through a general reduction, can tighten this gap further, to just $\log k$ for shared BSTs.

To our knowledge, no prior work proposes a general framework that converts an approximation algorithm for a single-objective (weighted-sum) problem into a provably fair multi-agent algorithm with an approximation guarantee independent of the number of agents (Chapter 4), nor studies the specialization of such a guarantee to a single shared data structure measured against k simultaneous personalized optima (Chapter 3). These are the two central contributions of this thesis.

Chapter 3

Multi-Agent Binary Search Trees

This chapter studies the multi-agent fairness problem introduced in Chapter 1 for a concrete and practically important shared resource: a single binary search tree accessed by k agents, each with its own access distribution. We work in the *deterministic* setting throughout: the resource shared by all k agents is a single, fixed BST, not a distribution over trees. The system must deploy one concrete structure: the same tree is used every time any agent looks up a key, so there is no room for a randomized decision rule at query time. This restriction turns out to be inexpensive in a formal sense, since any randomized BST can be converted into a deterministic one whose depth for every item grows by only a constant factor (Theorem 3.12). The lower bounds we prove in this chapter therefore apply asymptotically to randomized constructions, including the distributions over trees that the general framework of Chapter 4 would produce if applied directly to this problem. All logarithms are base 2 unless otherwise noted, and the root of a tree has depth 1.

Setup. We are given n items $x_1 < x_2 < \cdots < x_n$ and k agents. Agent i has an access distribution $p_i = (p_i(x_1), \ldots, p_i(x_n))$ over the items, where $p_i(x_j) \geq 0$ and $\sum_j p_i(x_j) = 1$. We call $\mathrm{supp}(p_i) = \{x_j : p_i(x_j) > 0\}$ the *support* of agent i; a collection of distributions has *pairwise disjoint supports* if $\mathrm{supp}(p_i) \cap \mathrm{supp}(p_{i'}) = \varnothing$ for $i \neq i'$. A binary search tree T on the n items: let $d_T(x_j)$ be the depth of item x_j in T; accessing x_j costs $d_T(x_j)$ comparisons.

For brevity, we henceforth write *tree* in place of *binary search tree*. The expected access cost of agent i under tree T is

$$\mathrm{cost}(T, p_i) \;=\; \sum_{j=1}^{n} p_i(x_j) \cdot d_T(x_j).$$

The goal is to find a single tree T that is simultaneously "good" for all k agents.

The benchmark. The natural benchmark for each agent is the cost of the optimal tree for that agent in isolation. For agent i, let $C_i = \min_T \mathrm{cost}(T, p_i)$ denote the optimal BST cost for p_i alone; more generally, for any distribution p we write $C(p) = \min_T \mathrm{cost}(T, p)$. When a single tree T is shared by all agents, the cost $\mathrm{cost}(T, p_i)$ will generally exceed C_i for every agent i: no single tree can simultaneously be optimal for two agents unless their distributions happen to agree. The question is how much overhead each agent is forced to bear. Section 3.1 collects the classical information-theoretic tools that let us answer this question precisely, most importantly a sharp, essentially tight characterization of C_i in terms of the Shannon entropy of p_i.

Two measures of overhead. We study two natural ways to measure how much worse than C_i any agent does under a shared tree T.

Definition 3.1 (Additive optimality)**.** The *additive overhead* of tree T for agent i is $\mathrm{cost}(T, p_i) - C_i$. A tree T is *α-additive-optimal* if

$$\max_{1 \le i \le k} \big(\mathrm{cost}(T, p_i) - C_i\big) \;\le\; \alpha.$$

Definition 3.2 (Multiplicative optimality)**.** The *multiplicative overhead* of tree T for agent i is the ratio $\mathrm{cost}(T, p_i)/C_i$. A tree T is *ρ-multiplicative-optimal* if

$$\max_{1 \le i \le k} \frac{\mathrm{cost}(T, p_i)}{C_i} \;\le\; \rho.$$

Both measures are meaningful, but they capture different aspects of fairness. Additive optimality treats all agents on the same absolute scale: an additive overhead of α means every agent pays at most α extra comparisons per access compared to their ideal tree. This is appropriate when the agents' optimal costs $C_1, \ldots, C_k$ are of comparable magnitude.

Multiplicative optimality is more appropriate when the agents' entropies, and hence their optimal costs, vary widely. An agent with a very concentrated distribution may have C_i close to 1 (nearly a single lookup), while another with a near-uniform distribution over n items has $C_i \approx \log n$. An additive bound of $\alpha = O(\log k)$ says nothing meaningful to the first agent: a shared tree might incur $\text{cost}(T, p_i) = 1 + O(\log k)$, which is several times C_i and represents a severe degradation. A multiplicative bound of ρ guarantees, regardless of the scale of C_i, that no agent ever pays more than ρ times their ideal cost.

Main results. The results of this chapter are summarized in Table 3.1. The parameter λ is the balancing parameter of Definition 3.17 and γ is the regret factor of Definition 3.21; both are instance-dependent quantities computed from the agents' distributions, and both satisfy $\gamma \leq \lambda = O(\log k)$ (Section 3.3.4).

Table 3.1: Personalized performance guarantees for a shared BST. The upper bounds hold simultaneously for every agent; the lower bounds state that every deterministic *or randomized* shared BST incurs the displayed cost for at least one agent.

Algorithm	Type	Guarantee for every agent i	Unavoidable for some agent i
Fuse	Additive	$C_i + \log k + \log\log n + O(1)$	$C_i + \log k - O(1)$
Comb	Multiplicative	$\lceil \log(k+1) \rceil \, C_i$	$(\log k - O(1))\, C_i$
Blend	Multiplicative	$\lambda \, C_i$	$(1 - o(1))\, \lambda \, C_i$
Tune	Multiplicative	$\gamma \, C_i$	$(1 - o(1))\, \gamma \, C_i$

For additive optimality (Section 3.2), a simple algorithm called Fuse, which averages all agents' distributions and builds a near-optimal BST for the average, achieves an additive overhead of $\log k + \log\log n + O(1)$, and we show that $\log k - O(1)$ is unavoidable in the worst case, so the dependence on k is essentially tight.

For multiplicative optimality (Section 3.3), we present three algorithms of increasing

sophistication. The first, COMB, a layer-by-layer construction that uses only the agents' individual optimal trees, achieves $\lceil \log(k+1) \rceil$ overhead and is worst-case optimal. The second, BLEND, introduces cost-sensitive mixture weights parameterized by λ; it achieves λ overhead, where $\lambda = O(\log k)$ but can be substantially smaller when agents have large personalized cost scales, and this bound is asymptotically tight when the agents' distributions have pairwise disjoint supports. The third, TUNE, optimizes over all possible combined distributions via a cross-entropy minimax criterion, achieving γ overhead with $\gamma \leq \lambda$; this is always at least as tight as BLEND, and the matching lower bound holds for arbitrary access distributions, without a disjoint-support assumption.

Finally, all four constructions can be made *robust* to inaccurate predictions: Section 3.4 shows how to guarantee worst-case depth $O(\log n)$ for every item, regardless of how the predicted distributions were obtained, while preserving each algorithm's original guarantee up to a vanishing additive correction.

The remainder of this chapter is organized as follows. Section 3.1 collects the classical and refined facts about single-distribution binary search trees that all of our constructions rely on, together with the derandomization result justifying our focus on deterministic trees. Section 3.2 presents the additive-optimality results. Section 3.3 presents the three multiplicative algorithms in order of increasing power, together with matching lower bounds, and concludes with a comparison of the three algorithms. Section 3.4 shows how to make every construction robust to inaccurate predictions.

3.1 Preliminaries

All of our constructions reduce the multi-agent problem to building a single tree for a cleverly chosen distribution, so we first record the facts that govern how well one tree can serve one distribution. These concern an ordinary single-distribution BST: a tree T on the n items, a probability distribution p over them, the expected access cost $\mathrm{cost}(T, p) = \sum_j p(x_j)\, d_T(x_j)$,

and the optimal (static) cost $C(p) = \min_T \mathrm{cost}(T, p)$.

Entropy, cross-entropy, and divergence. The *entropy* of a distribution p is $H(p) := -\sum_x p(x) \log p(x)$, with the convention $0 \log 0 = 0$; for agent i we write $H_i := H(p_i)$. For two distributions q and p, their *cross-entropy* is $H(q, p) := -\sum_x q(x) \log p(x)$; informally, it reflects the cost of using a structure designed according to p on accesses drawn from q. If $q(x) > 0$ and $p(x) = 0$ for some item x, then $H(q, p) = +\infty$. The *Kullback–Leibler divergence* from q to p is

$$D_{\mathrm{KL}}(q \,\|\, p) \;:=\; \sum_x q(x) \log \frac{q(x)}{p(x)},$$

where the double bar $\|$ is the standard notation for this and other divergences, used instead of a comma to emphasize that, unlike a distance, D_{KL} is not symmetric in its two arguments; this measures the mismatch between q and p. The two quantities are related by $H(q, p) = H(q) + D_{\mathrm{KL}}(q \,\|\, p)$. Cross-entropy is central to our analysis because most of our algorithms construct a tree from a single common reference distribution and then evaluate its cost under each agent's access distribution.

Static optimality: a sharp entropy characterization. Classical results relate the optimal BST cost $C(p)$ to the entropy $H(p)$. The cost is essentially the entropy of the access distribution, up to a small additive slack in one direction and an implicit but tightly-characterized function of the entropy in the other.

Definition 3.3 (Implicit entropy lower bound)**.** For $z > 1$, define

$$F(z) \;:=\; z + \log z + (z - 1) \log\left(1 + \frac{1}{z - 1}\right) - 1. \tag{3.1}$$

The function F is strictly increasing on $(1, \infty)$. For a distribution p, define $L(p) := F^{-1}(H(p))$ when $H(p) > 0$, and $L(p) := 1$ when $H(p) = 0$. For agent i, we write $L_i := L(p_i)$. Intuitively, F is the tightest relationship between $H(p)$ and $C(p)$ obtainable from Mehlhorn's classical

lower-bound family (Lemma 3.4).

The lower bound on $C(p)$ we are after requires more care than the upper bound, since $C(p)$ is a genuine minimum over all trees. We derive it from a classical family of lower bounds due to Mehlhorn.

Lemma 3.4 (Mehlhorn [Mehlhorn, 1984])**.** *For every distribution p and every real number d,*

$$C(p) \;\geq\; \frac{H(p) - d}{\log(2 + 2^{-d})}.$$

Optimizing the free parameter d in Lemma 3.4 for a fixed target value of $C(p)$ yields the following relationship between $H(p)$ and $C(p)$.

Lemma 3.5. *For every distribution p, the optimal BST cost $C(p)$ satisfies*

$$H(p) \;\leq\; C(p) + \log C(p) + \big(C(p) - 1\big) \log\left(1 + \frac{1}{C(p) - 1}\right) - 1.$$

Proof. If $C(p) = 1$, then p is concentrated on a single item and $H(p) = 0$, so the claim holds trivially. Assume therefore that $C(p) > 1$ and set $d = \log(C(p) - 1) - 1$ in Lemma 3.4. Rearranging the resulting inequality gives the claim. □

With this in hand, $L(p)$ depends only on the entropy of p, lower-bounds its optimal BST cost, and in fact tracks $C(p)$ closely, as the following lemma makes precise.

Lemma 3.6 (Static-optimality bounds [Mehlhorn, 1975, 1984])**.** *For every distribution p over n ordered items,*

$$L(p) \;\leq\; C(p) \;\leq\; H(p) + 1 \;\leq\; L(p) + \log L(p) + 1.45. \tag{3.2}$$

Moreover,

$$L(p) \;\geq\; H(p) - \log\big(H(p) + 1\big) - 0.45, \tag{3.3}$$

$$H(p) \;\le\; 1.59\,L(p). \tag{3.4}$$

Consequently, since $H(p) \le \log n$,

$$C(p) \;\ge\; H(p) - \log(\log n + 1) - 0.45. \tag{3.5}$$

The upper bound $C(p) \le H(p) + 1$ is achieved by an explicit construction (stated below as Lemma 3.7) that is considerably stronger than the average-cost guarantee suggests: it controls the depth of *every* item individually, not merely the average depth. The lower bound follows from Lemma 3.5 above.

Proof of Lemma 3.6. The upper bound $C(p) \le H(p) + 1$ follows from Lemma 3.7 below, summing its per-item guarantee against p itself. For $C(p) > 1$, the right-hand side of Lemma 3.5 is exactly $F(C(p))$ from (3.1), and since F is increasing on $(1, \infty)$ it is invertible there; $F(C(p)) \ge H(p) = F(L(p))$ therefore implies $C(p) \ge L(p)$, which also holds trivially when $C(p) = 1$. While $L(p)$ has no closed form, it admits explicit bounds. Since $x \log(1 + 1/x) \le \log e \le 1.45$ for all $x > 0$,

$$H(p) + 1 \;=\; F(L(p)) + 1 \;\le\; L(p) + \log L(p) + 1.45,$$

which is the remaining inequality of (3.2). For (3.3), applying the same bound directly to $H(p) = F(L(p)) = L(p) + \log L(p) + (L(p) - 1)\log\bigl(1 + \frac{1}{L(p)-1}\bigr) - 1$ gives $H(p) \le L(p) + \log L(p) + 0.45$, i.e., $L(p) \ge H(p) - \log L(p) - 0.45$. Since $L(p) \le C(p) \le H(p) + 1$ by (3.2), this in turn gives $L(p) \ge H(p) - \log(H(p) + 1) - 0.45$. For (3.4), the line $x \log 3$ touches the curve $F(x)$ at exactly $x = 3$ and lies above it everywhere else, so $H(p) = F(L(p)) \le (\log 3) \cdot L(p) \le 1.59\,L(p)$. Finally, since $H(p) \le \log n$ always, applying (3.3) together with $C(p) \ge L(p)$ gives $C(p) \ge H(p) - \log(\log n + 1) - 0.45$, proving (3.5). □

The exact value of $C(p)$, together with an optimal tree, can be computed in $O(n^2)$ time using Knuth's dynamic program [Knuth, 1971]. Our upper bounds instead use the following

near-optimal construction, which can be computed in $O(n \log n)$ time.

Lemma 3.7 (Mehlhorn tree [Mehlhorn, 1975])**.** *For every distribution p over X, there exists a tree T_p such that*

$$d_{T_p}(x) \;\leq\; -\log p(x) + 1$$

for every item x with $p(x) > 0$.

The strength of Lemma 3.7 is that it bounds the depth of each item tightly in terms of its own log-probability $-\log p(x)$ (the information-theoretic ideal), and does so for every item *simultaneously* with an explicit additive constant. The optimal BST, by contrast, minimizes only the *average* depth. Summing the per-item bound of Lemma 3.7 against a possibly different distribution q yields a cross-entropy bound, the form used throughout this chapter.

Corollary 3.8 (Cross-entropy guarantee)**.** *For any distributions p and q, the Mehlhorn tree T_p satisfies*

$$\mathrm{cost}(T_p, q) \;\leq\; H(q, p) + 1 \;=\; H(q) + D_{\mathrm{KL}}(q \,\|\, p) + 1.$$

In particular, $\mathrm{cost}(T_p, p) \leq H(p) + 1$.

Corollary 3.8 is the main bridge between distributions and shared BSTs: once a common reference distribution p is chosen, the cost of agent i is controlled by the cross-entropy $H(p_i, p)$. Our algorithms differ primarily in how they choose this reference distribution.

Depth robustness. The bounds above assume the supplied distributions accurately describe future accesses. When they are instead estimated or predicted, an item assigned an extremely small predicted probability may be placed arbitrarily deep by Lemma 3.7, even if it is frequently accessed in the true workload. We therefore also consider a structural worst-case guarantee, independent of prediction quality.

Definition 3.9 (Depth robustness)**.** A construction is *ϕ-depth-robust* if its output tree T satisfies $\max_{x \in X} d_T(x) \leq \phi$ for every collection of input distributions.

Section 3.4 shows how to make every algorithm in this chapter ϕ-depth-robust with $\phi = 2\log n + O(1)$, which is optimal up to a factor of two: any binary tree on n items has maximum depth at least $\lceil \log(n+1) \rceil$.

Randomized BSTs and derandomization. A *randomized* BST $\overline{T}$ is a probability distribution over deterministic trees. Its cost for a distribution p is $\mathrm{cost}(\overline{T}, p) := \mathbb{E}_{T\sim\overline{T}}[\mathrm{cost}(T, p)]$; equivalently, for an item x, writing $\overline{d}(x) := \mathbb{E}_{T\sim\overline{T}}[d_T(x)]$ for its expected depth, we have $\mathrm{cost}(\overline{T}, p) = \sum_x p(x)\,\overline{d}(x)$. A search in a tree proceeds by a three-way comparison at each node (less-than, equal, or greater-than), so each item corresponds to a codeword over a ternary alphabet whose length is its depth, and these codewords form a prefix-free set. This yields a base-3 Kraft inequality, the standard Kraft inequality [Cover and Thomas, 2006] adapted from a binary to a ternary alphabet.

Lemma 3.10 (Base-3 Kraft inequality)**.** *For any tree T on n items, the depths satisfy $\sum_{j=1}^{n} 3^{-d_T(x_j)} \le 1$.*

The following lemma extends this to randomized trees, and is the bridge between randomized and deterministic BSTs that we use to prove the derandomization theorem below.

Lemma 3.11 (Kraft inequality for randomized BSTs)**.** *Let $\overline{T}$ be any randomized BST on the n items, and let $d_i := \mathbb{E}_{T\sim\overline{T}}[d_T(x_i)]$. Then $\sum_{i=1}^{n} 3^{-d_i} \le 1$.*

Proof. Since $x \mapsto 3^{-x}$ is convex, Jensen's inequality gives $3^{-d_i} = 3^{-\mathbb{E}_{T\sim\overline{T}}[d_T(x_i)]} \le \mathbb{E}_{T\sim\overline{T}}[3^{-d_T(x_i)}]$. Summing over i and applying Lemma 3.10 to each realization $T \sim \overline{T}$,

$$\sum_i 3^{-d_i} \;\le\; \sum_i \mathbb{E}_{T\sim\overline{T}}\big[3^{-d_T(x_i)}\big] \;=\; \mathbb{E}_{T\sim\overline{T}}\Big[\sum_i 3^{-d_T(x_i)}\Big] \;\le\; 1. \qquad \square$$

Theorem 3.12 (Derandomization)**.** *Let $\overline{T}$ be any randomized BST, and let $\overline{d}(x) := \mathbb{E}_{T\sim\overline{T}}[d_T(x)]$. There exists a deterministic tree T' such that, for every item x,*

$$d_{T'}(x) \;\le\; (\log 3)\,\overline{d}(x) + 1 \;<\; 1.59\,\overline{d}(x) + 1.$$

Consequently, for every access distribution p, $\mathrm{cost}(T', p) \le 1.59\,\mathrm{cost}(\overline{T}, p) + 1$.

Proof. By Lemma 3.11, $Z := \sum_i 3^{-\overline{d}(x_i)} \le 1$. Let $q(x_i) := 3^{-\overline{d}(x_i)}/Z$, a probability distribution over the items. Applying Lemma 3.7 to q yields a deterministic tree T' with

$$d_{T'}(x_i) \;\le\; -\log q(x_i) + 1 \;=\; \overline{d}(x_i)\log 3 + \log Z + 1 \;\le\; 1.59\,\overline{d}(x_i) + 1,$$

using $\log Z \le 0$ since $Z \le 1$. The cost bound follows by multiplying by $p(x_i)$ and summing over all items. □

Thus, randomization can improve the personalized objectives of this chapter by at most a constant factor. This justifies our focus on constructing deterministic shared trees, while also allowing every lower bound we prove to apply to the more general class of randomized BSTs.

3.2 Additive Optimality

The most natural approach to building a shared tree is to treat all agents symmetrically and optimize for their collective access pattern. Rather than favoring any single agent, we aggregate all distributions into one by taking their uniform average, and then build a near-optimal BST for that average.

Algorithm Fuse. Let $p_{\mathrm{avg}} = \frac{1}{k}\sum_{i=1}^{k} p_i$ be the uniform average of all agents' distributions. Fuse applies the Mehlhorn tree construction of Lemma 3.7 to p_{avg}, and returns the resulting tree T_{avg}. The distribution p_{avg} can be computed in $O(nk)$ time, giving a total running time of $O(nk + n\log n)$.

The key observation is that every agent contributes a $1/k$ fraction of the mixture, so $p_{\mathrm{avg}}(x) \ge p_i(x)/k$ for every item x: using the shared mixture increases each item's ideal log-probability by at most $\log k$.

Theorem 3.13 (Additive upper bound)**.** *The tree T_{avg} is $\big(\log k + \log(\log n + 1) + 1.45\big)$-additive-optimal. That is, for every agent i,*

$$\text{cost}(T_{\text{avg}}, p_i) - C_i \;\le\; \log k + \log(\log n + 1) + 1.45.$$

Proof. Since $p_{\text{avg}}(x) \ge p_i(x)/k$ for every $x \in \text{supp}(p_i)$, $H(p_i, p_{\text{avg}}) \le H_i + \log k$. Corollary 3.8 therefore gives

$$\text{cost}(T_{\text{avg}}, p_i) \;\le\; H_i + \log k + 1.$$

By Lemma 3.6, $C_i \ge H_i - \log(\log n + 1) - 0.45$. Subtracting the two inequalities gives

$$\text{cost}(T_{\text{avg}}, p_i) - C_i \;\le\; \log k + \log(\log n + 1) + 1.45. \qquad \square$$

Thus, sharing costs each agent only about one additional comparison whenever the number of agents doubles. This logarithmic dependence on k is unavoidable:

Theorem 3.14 (Additive lower bound)**.** *For every k, there is an instance with k agents such that every deterministic or randomized tree T satisfies*

$$\max_{1 \le i \le k} \big(\text{cost}(T, p_i) - C_i\big) \;\ge\; \log k - 2.$$

Proof. Fix any integers $k, m \ge 1$ and let $n = km$. Partition the n items into k consecutive groups of m items each. Give agent i the uniform distribution over its group: $p_i(x_j) = 1/m$ for x_j in group i, and zero otherwise. The supports are pairwise disjoint by construction, and $H_i = \log m$ for every i.

The average distribution $p_{\text{avg}} = \frac{1}{k}\sum_{i=1}^{k} p_i$ is uniform over all $n = km$ items, so $H(p_{\text{avg}}) = \log(km) = \log k + \log m$. Since $\text{cost}(T, \cdot)$ is linear in the distribution, the average agent cost

under any tree T equals the cost of p_{avg} under T, which is the average depth of all items in T:

$$\frac{1}{k}\sum_{i=1}^{k}\text{cost}(T,p_i) \;=\; \text{cost}(T,p_{\text{avg}}) \;=\; \frac{1}{n}\sum_{j=1}^{n}d_T(x_j).$$

Any *deterministic* tree on n items has average depth at least $C(p_{\text{avg}})$, since $C(p_{\text{avg}})$ is by definition the minimum average depth over all trees on n items. Since p_{avg} is uniform, this minimum is achieved by a balanced, nearly complete binary tree, for which a direct count of depths gives the classical bound $C(p_{\text{avg}}) \geq \log n - 1$. Because this inequality holds for every deterministic tree, taking an expectation over any distribution of trees gives the same bound for randomized trees. Hence

$$\frac{1}{k}\sum_{i=1}^{k}\text{cost}(T,p_i) \;\geq\; \log k + \log m - 1.$$

Each p_i is itself uniform over its group of m items, so the same balanced-tree bound applies directly: exhibiting the balanced tree gives $C_i \leq \log m + 1$ for every i. Therefore the average additive overhead satisfies

$$\frac{1}{k}\sum_{i=1}^{k}\big(\text{cost}(T,p_i) - C_i\big) \;\geq\; \log k - 2.$$

By an averaging argument we have at least one agent i^* has additive overhead at least $\log k - 2$. □

The upper and lower bounds match in their leading $\log k$ term. The remaining $O(\log\log n)$ gap in the upper bound arises from the entropy-based analysis (specifically from the gap between H_i and C_i recorded in (3.5)), and is not an artifact of FUSE specifically: since C_i can be as small as $H_i - \log(\log n + 1) - 0.45$, any analysis that passes through H_i must incur this term, regardless of the algorithm being analyzed.

3.3 Multiplicative Optimality

We now turn to multiplicative optimality, which gives meaningful guarantees even when agents' optimal costs vary widely. We present three algorithms in order of increasing sophistication, each motivated by a limitation of the previous one. COMB uses only the agents' individual optimal trees, without requiring their underlying access distributions, and achieves a worst-case-optimal structural guarantee. BLEND uses entropy-based estimates of the agents' personalized costs to obtain a simple and efficient instance-sensitive guarantee, i.e., one that improves automatically on particular instances rather than only matching the worst case. TUNE additionally exploits overlap among the agents' distributions and yields the strongest instance-specific bound, at the cost of solving a convex optimization problem.

3.3.1 Layer-by-Layer Construction: COMB

The first algorithm is motivated by a simple question: is there a way to build a shared tree that directly respects the structure of every agent's optimal tree, rather than ignoring them entirely? The answer is yes. Each agent's optimal tree T_i^* encodes a priority ordering over items: items near the root are the most frequently accessed. The COMB algorithm extracts one representative item per agent per interval (their LCA), arranges these representatives in a balanced BST, and recurses. This ensures that the depth of any item in the shared tree is within a $\log k$ factor of its depth in any agent's optimal tree, and requires neither the access probabilities nor an optimization over mixture distributions once the individual trees are available.

Algorithm. Let T_i^* denote an optimal BST for agent i. For an interval (l, r) of items, define $\mathrm{LCA}(T_i^*, l, r)$ as the lowest common ancestor in T_i^* of all items in (l, r); intuitively, this is the root of the minimal subtree of T_i^* that contains all items in the interval.

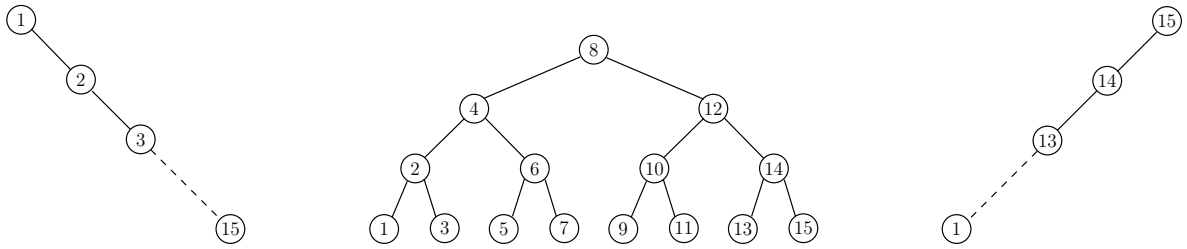
The algorithm, which we call COMB, is defined recursively on an interval (l, r), initially called with $(l, r) = (-\infty, +\infty)$.

Algorithm 1 COMB(l, r)

if (l, r) contains no items **then return None**
end if
for $i = 1$ to k **do**
    $a_i \leftarrow \mathrm{LCA}(T_i^*,\, l,\, r)$
end for
$T \leftarrow$ perfect BST on the distinct representatives among $\{a_1, \ldots, a_k\}$
for each empty subtree slot c of T **do**
    $(l', r') \leftarrow$ interval of items mapped to slot c
    attach COMB(l', r') at slot c
end for
return T

At each recursive call, the algorithm collects one representative item per agent (the LCA of the current interval in that agent's optimal tree) and arranges these k representatives in a perfect balanced BST. It then recurses independently on each sub-interval induced by the leaves of that balanced BST.

Figures 3.1 and 3.2 illustrate COMB for $k = 3$ agents and $n = 15$ items. Figure 3.1 shows the three input optimal trees T_1^*, T_2^*, T_3^*; Figure 3.2 shows the resulting combined tree.



(a) T_1^* (b) T_2^* (c) T_3^*

Figure 3.1: The three input optimal trees for the COMB example with $k = 3$ agents and $n = 15$ items. Agents A_1 and A_3 concentrate their accesses on the two extreme items, so their optimal trees are linear chains rooted at item 1 and item 15 respectively; the dashed edges indicate that each chain continues through the remaining items. Agent A_2 accesses all items roughly uniformly, so its optimal tree is the balanced tree rooted at the median item 8.

Theorem 3.15 (COMB upper bound)**.** *For every agent i and every item x,*

$$d_{T_{\mathrm{comb}}}(x) \;\le\; \lceil \log(k+1) \rceil \cdot d_{T_i^*}(x).$$

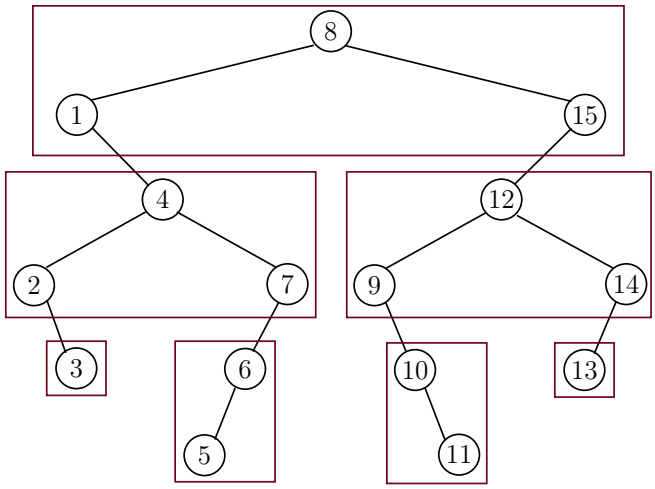


Figure 3.2: Result of COMB on the three trees of Figure 3.1. Each red box marks the perfect BST constructed during one recursive call. The top box covers the full interval $(1, 15)$ and collects each agent's LCA of that interval: item 8 (the root of T_2^*), item 1 (the root of T_1^*), and item 15 (the root of T_3^*); these three are arranged in a perfect BST. The algorithm then recurses on the sub-intervals $[1, 7]$ and $[9, 15]$, whose representatives are $\{2, 4, 7\}$ and $\{9, 12, 14\}$ and which produce the two middle boxes, continuing until every item is placed.

Consequently, $\mathrm{cost}(T_{\mathrm{comb}}, p_i) \le \lceil \log(k+1) \rceil \cdot \mathrm{cost}(T_i^*, p_i)$*; in particular, if* T_i^* *is optimal for* p_i*, the tree* T_{comb} *is* $\lceil \log(k+1) \rceil$*-multiplicative-optimal.*

Proof. Fix an item x_j and an agent i. Consider the sequence of recursive calls that ultimately place x_j in T_{comb}. There is a chain of nested intervals $(l_1, r_1) \supset (l_2, r_2) \supset \cdots \supset (l_s, r_s)$ such that x_j first appears as a representative at level s. At each level t, the representative $a_t = \mathrm{LCA}(T_i^*, l_t, r_t)$ is placed in a balanced BST of at most k items, contributing at most $\lceil \log(k+1) \rceil$ to the depth of x_j. Because a_t was already placed in the balanced BST built at level t, it is removed from consideration in the recursive calls below it, so $a_t \notin (l_{t+1}, r_{t+1})$. The elements $a_1, \ldots, a_s$ are therefore distinct ancestors of x_j in T_i^*, giving $s \le d_{T_i^*}(x_j)$. Therefore

$$d_{T_{\mathrm{comb}}}(x_j) \;\le\; s \cdot \lceil \log(k+1) \rceil \;\le\; d_{T_i^*}(x_j) \cdot \lceil \log(k+1) \rceil,$$

which proves the claim. Multiplying by $p_i(x_j)$ and summing over all items,

$$\mathrm{cost}(T_{\mathrm{comb}}, p_i) \;\le\; \lceil \log(k+1) \rceil \cdot \mathrm{cost}(T_i^*, p_i). \qquad \square$$

Exact individual trees T_i^* can be computed using Knuth's dynamic program. Alternatively, one may apply COMB to faster near-optimal trees, in which case the theorem compares its cost with those input trees rather than directly with C_i.

Theorem 3.16 (Worst-case lower bound)**.** *For every k, there is a family of k distributions such that every deterministic or randomized tree T satisfies*

$$\max_{1 \le i \le k} \frac{\mathrm{cost}(T, p_i)}{C_i} \;\ge\; \log k - 1.$$

Proof. Consider k agents where agent i accesses only item x_i: $p_i(x_i) = 1$ and $p_i(x_j) = 0$ for $j \neq i$. The optimal tree for agent i places x_i at the root, giving $C_i = 1$, so agent i's multiplicative overhead under T is simply $\mathbb{E}_T[d_T(x_i)]$.

For every deterministic realization of T on $x_1 < \cdots < x_k$, the quantity $\frac{1}{k}\sum_i d_T(x_i)$ is the average depth of T on these k items, which is at least $\log k - 1$, exactly as in the proof of Theorem 3.14. Averaging over agents and then over the randomness of T,

$$\max_i \mathbb{E}_T[d_T(x_i)] \;\ge\; \frac{1}{k}\sum_i \mathbb{E}_T[d_T(x_i)] \;=\; \mathbb{E}_T\Big[\tfrac{1}{k}\sum_i d_T(x_i)\Big] \;\ge\; \log k - 1,$$

so at least one agent has multiplicative overhead at least $\log k - 1$, whether T is deterministic or randomized. □

3.3.2 Entropy-Weighted Construction: BLEND

The worst-case factor of COMB depends only on k and can be pessimistic when the agents' personalized costs are large: such agents can tolerate a larger absolute increase while still receiving a small multiplicative overhead, but COMB ignores this asymmetry. Our next algorithm, BLEND, captures it through cost-sensitive mixture weights: agents with a larger cost scale L_i receive less weight in the combined distribution, reflecting the fact that a generic tree already does reasonably well for them.

Definition 3.17 (Balancing parameter and BLEND)**.** Recalling L_i from Definition 3.3, let λ be the unique positive real number satisfying

$$\sum_{i=1}^{k} 2^{\,H_i+1-\lambda L_i} \;=\; 1, \tag{3.6}$$

and define the weight of agent i as $w_i := 2^{\,H_i+1-\lambda L_i}$; by (3.6) the weights sum to one. The weights are chosen deliberately so that, when Lemma 3.7 is applied to p_{blend}, agent i's cost works out to exactly λL_i (Theorem 3.18 below). BLEND forms the mixture distribution

$$p_{\text{blend}} \;:=\; \sum_{i=1}^{k} w_i\, p_i,$$

and returns the Mehlhorn tree T_{blend} of Lemma 3.7 applied to p_{blend}.

Such a λ exists and is unique because the left-hand side of (3.6) is a continuous, strictly decreasing function of λ that tends to $+\infty$ as $\lambda \to -\infty$ and to 0 as $\lambda \to \infty$. Since $H_i \geq 0$ for every i, at $\lambda = 0$ the left-hand side is at least $2k \geq 2 > 1$, so the unique root is positive. The parameter λ balances the agents according to their individual cost scales, and can be approximated by a one-dimensional binary search, making BLEND simple and efficient to implement.

Theorem 3.18 (BLEND upper bound)**.** *The tree T_{blend} satisfies $\text{cost}(T_{\text{blend}}, p_i) \leq \lambda\, C_i$ for every agent i; in particular, T_{blend} is λ-multiplicative-optimal.*

Proof. Since $p_{\text{blend}}(x_j) = \sum_\ell w_\ell\, p_\ell(x_j) \geq w_i\, p_i(x_j) = 2^{\,H_i+1-\lambda L_i}\, p_i(x_j)$, taking logarithms gives

$$-\log p_{\text{blend}}(x_j) \;\leq\; -\log p_i(x_j) - H_i - 1 + \lambda L_i.$$

By Lemma 3.7, $d_{T_{\text{blend}}}(x_j) \leq -\log p_{\text{blend}}(x_j) + 1$, so

$$d_{T_{\text{blend}}}(x_j) \;\leq\; -\log p_i(x_j) - H_i + \lambda L_i.$$

Multiplying by $p_i(x_j)$ and summing over all items,

$$\operatorname{cost}(T_{\text{blend}}, p_i) \;\le\; H_i - H_i + \lambda L_i \;=\; \lambda L_i.$$

By Lemma 3.6, $C_i \ge L_i$, so $\operatorname{cost}(T_{\text{blend}}, p_i)/C_i \le \lambda L_i/C_i \le \lambda$. □

Proposition 3.19 (Upper bound on λ)**.**

$$\lambda \;\le\; 1.59 + \frac{1 + \log k}{\min_{1 \le i \le k} L_i}.$$

Proof. Since $\sum_{i=1}^{k} 2^{\,H_i + 1 - \lambda L_i} = 1$, by an averaging argument we have an agent i with $2^{\,H_i + 1 - \lambda L_i} \ge 1/k$, so $\lambda \le (H_i + 1 + \log k)/L_i$. Since $H_i \le 1.59\, L_i$ for every i by (3.4), this is at most $1.59 + (1 + \log k)/L_i \le 1.59 + (1 + \log k)/\min_{1 \le i \le k} L_i$. □

Thus Blend retains the same $O(\log k)$ worst-case dependence as Comb, while its factor decreases as the smallest agent cost scale $L_{\min} := \min_{1 \le i \le k} L_i$ increases. The next theorem shows that the dependence on λ is asymptotically tight when the agents' distributions have pairwise disjoint supports.

Theorem 3.20 (Tightness of Blend)**.** *For agents with pairwise disjoint supports, every deterministic or randomized tree T incurs multiplicative overhead at least*

$$\lambda \;-\; O\!\left(\frac{\lambda\big(1 + \log E_{\text{blend}}\big)}{E_{\text{blend}}}\right), \qquad \text{where } E_{\text{blend}} := \sum_{i=1}^{k} w_i L_i.$$

Consequently, the lower bound is $(1 - o(1))\lambda$ as E_{blend} grows.

Proof. We first compute the entropy of the mixture distribution. Since the supports of the p_i are pairwise disjoint, every item in $\operatorname{supp}(p_i)$ has probability $w_i p_i(x)$ under p_{blend}, so

$$H(p_{\text{blend}}) \;=\; -\sum_{i=1}^{k} \sum_{x \in \operatorname{supp}(p_i)} w_i p_i(x) \log\big(w_i p_i(x)\big) \;=\; \sum_{i=1}^{k} w_i H_i - \sum_{i=1}^{k} w_i \log w_i.$$

By definition $w_i = 2^{H_i+1-\lambda L_i}$, so $-\log w_i = -H_i - 1 + \lambda L_i$. Substituting this and using $\sum_i w_i = 1$ gives

$$H(p_{\text{blend}}) \;=\; \sum_{i=1}^{k} w_i H_i + \sum_{i=1}^{k} w_i(-H_i - 1 + \lambda L_i) \;=\; \lambda E_{\text{blend}} - 1. \tag{3.7}$$

Now let $\mathcal{T}$ be any deterministic or randomized tree, and write

$$R := \max_{1 \le i \le k} \operatorname{cost}(\mathcal{T}, p_i)/C_i,$$

where, for a randomized tree, $\operatorname{cost}(\mathcal{T}, p_i)$ denotes its expected cost. By linearity,

$$\sum_{i=1}^{k} w_i \operatorname{cost}(\mathcal{T}, p_i) \;=\; \operatorname{cost}(\mathcal{T}, p_{\text{blend}}).$$

Every deterministic realization has cost at least $C(p_{\text{blend}})$ on p_{blend}; consequently, the same lower bound holds in expectation for a randomized tree, so, by (3.2),

$$\sum_{i=1}^{k} w_i \operatorname{cost}(\mathcal{T}, p_i) \;\ge\; C(p_{\text{blend}}) \;\ge\; L(p_{\text{blend}}).$$

On the other hand, by definition of R, $\sum_{i=1}^{k} w_i \operatorname{cost}(\mathcal{T}, p_i) \le R \sum_{i=1}^{k} w_i C_i$, so

$$R \;\ge\; \frac{L(p_{\text{blend}})}{\sum_{i=1}^{k} w_i C_i}.$$

We bound the numerator and denominator separately. By (3.3) and (3.7),

$$L(p_{\text{blend}}) \;\ge\; H(p_{\text{blend}}) - \log\big(H(p_{\text{blend}}) + 1\big) - 0.45 \;=\; \lambda E_{\text{blend}} - \log(\lambda E_{\text{blend}}) - 1.45.$$

Moreover, using $C_i \le H_i + 1$ and $H_i + 1 \le L_i + \log L_i + 1.45$ from (3.2),

$$\begin{aligned}\sum_{i=1}^{k} w_i C_i \;&\le\; \sum_{i=1}^{k} w_i\big(L_i + \log L_i + 1.45\big)\\ &=\; E_{\text{blend}} + \sum_{i=1}^{k} w_i \log L_i + 1.45\\ &\le\; E_{\text{blend}} + \log E_{\text{blend}} + 1.45,\end{aligned}$$

where the last step follows from the concavity of the logarithm. Therefore

$$R \;\ge\; \frac{\lambda E_{\text{blend}} - \log(\lambda E_{\text{blend}}) - 1.45}{E_{\text{blend}} + \log E_{\text{blend}} + 1.45}.$$

Finally,

$$\lambda - R \;\le\; \frac{\lambda\big(\log E_{\text{blend}} + 1.45\big) + \log(\lambda E_{\text{blend}}) + 1.45}{E_{\text{blend}} + \log E_{\text{blend}} + 1.45} \;=\; O\!\left(\frac{\lambda(1 + \log E_{\text{blend}})}{E_{\text{blend}}}\right),$$

so $R \ge \lambda - O\big(\lambda(1 + \log E_{\text{blend}})/E_{\text{blend}}\big) = (1 - o(1))\lambda$ as $E_{\text{blend}} \to \infty$. □

3.3.3 Cross-Entropy-Optimal Construction: TUNE

BLEND uses a specific parametric family of weights (exponential in entropy) that happens to be asymptotically tight when the agents' supports are disjoint. But there is no reason to restrict ourselves to this family: even when the agents' workloads overlap substantially, a single well-chosen reference distribution can approximate all of them simultaneously, and a fixed weighting rule need not find it. Our next algorithm, TUNE, optimizes the reference distribution directly.

For a candidate reference distribution p, Corollary 3.8 implies that the Mehlhorn tree T_p satisfies $\text{cost}(T_p, p_i) \le H(p_i, p) + 1$. Since $L_i \le C_i$ by Lemma 3.6, the multiplicative overhead of T_p for agent i is at most $\big(H(p_i, p) + 1\big)/L_i$. This motivates the following minimax parameter, chosen so that no agent suffers a disproportionately large such ratio.

Definition 3.21 (Regret factor and TUNE)**.** Define

$$\gamma \;:=\; \min_{p} \max_{1 \le i \le k} \frac{H(p_i, p) + 1}{L_i},$$

where the minimum ranges over all probability distributions p on X. Let p_{tune} be a minimizing distribution. TUNE returns the Mehlhorn tree T_{tune} for p_{tune}.

This is a convex optimization problem in p, admitting the saddle-point formulation

$$\gamma \;=\; \min_{p} \max_{\boldsymbol{\mu} \in \Delta_k} \sum_{i=1}^{k} \mu_i \, \frac{H(p_i, p) + 1}{L_i},$$

where Δ_k is the probability simplex over the k agents. Such problems can be solved using a Blahut–Arimoto-style alternating iteration [Blahut, 1972, Arimoto, 1972] or gradient-based methods such as projected gradient descent and subgradient methods [Boyd and Vandenberghe, 2004]; each iteration requires $O(nk)$ work, after which the Mehlhorn tree is constructed in $O(n \log n)$ time.

Theorem 3.22 (TUNE upper bound)**.** *The tree T_{tune} satisfies $\operatorname{cost}(T_{\text{tune}}, p_i) \le \gamma \, C_i$ for every agent i.*

Proof. By Lemma 3.7 applied to p_{tune}, summing the per-item bound against p_i gives $\operatorname{cost}(T_{\text{tune}}, p_i) \;\le\; H(p_i, p_{\text{tune}}) + 1$. By definition of γ, $\big(H(p_i, p_{\text{tune}}) + 1\big)/L_i \;\le\; \gamma$, so $\operatorname{cost}(T_{\text{tune}}, p_i) \le \gamma L_i$. Since $C_i \ge L_i$ by Lemma 3.6, $\operatorname{cost}(T_{\text{tune}}, p_i)/C_i \le \gamma L_i / C_i \le \gamma$. □

Unlike the tightness result for BLEND, the corresponding lower bound for TUNE holds without any disjoint-support assumption.

Theorem 3.23 (Tightness of TUNE)**.** *For arbitrary access distributions, every deterministic or randomized tree incurs multiplicative overhead at least*

$$\gamma \;-\; O\!\left(\frac{\gamma\big(1 + \log E_{\text{tune}}\big)}{E_{\text{tune}}} \right),$$

where E_{tune} is a weighted personalized cost scale defined in the proof below. Consequently, the lower bound is $(1 - o(1))\gamma$ as E_{tune} grows.

Proof. We first characterize p_{tune}. Write the minimax problem defining γ as

$$\gamma \;=\; \min_{p} \max_{\boldsymbol{\mu} \in \Delta_k} \sum_{i=1}^{k} \mu_i \, \frac{H(p_i, p) + 1}{L_i}.$$

By the minimax theorem (the objective is convex in p and linear in $\boldsymbol{\mu}$, and Δ_k is compact), the two optimizations can be exchanged:

$$\gamma \;=\; \max_{\boldsymbol{\mu} \in \Delta_k} \min_{p} \sum_{i=1}^{k} \mu_i \, \frac{H(p_i, p) + 1}{L_i}.$$

Let $\boldsymbol{\mu}^*$ be an optimal dual solution, and define $c_i := \mu_i^*/L_i$, $S := \sum_{i=1}^k c_i$, and $w_i^* := c_i/S$; these weights are nonnegative and sum to one. Let $\bar{p} := \sum_{i=1}^k w_i^* p_i$. Since cross-entropy is linear in its first argument, $\sum_{i=1}^k c_i H(p_i, p) = S\, H(\bar{p}, p)$, so

$$\sum_{i=1}^{k} \mu_i^* \frac{H(p_i, p) + 1}{L_i} \;=\; S\big(H(\bar{p}, p) + 1\big).$$

The cross-entropy $H(\bar{p}, p)$ is minimized over p at $p = \bar{p}$, so we may take $p_{\text{tune}} = \bar{p} = \sum_{i=1}^k w_i^* p_i$. Let $E_{\text{tune}} := \sum_{i=1}^k w_i^* L_i$. Because $\sum_i \mu_i^* = 1$ and $\mu_i^* = S\, w_i^* L_i$, we have $1 = S \sum_i w_i^* L_i = S\, E_{\text{tune}}$, so $S = 1/E_{\text{tune}}$. Evaluating the saddle-point objective at p_{tune} gives

$$\gamma \;=\; S\big(H(p_{\text{tune}}) + 1\big) \;=\; \frac{H(p_{\text{tune}}) + 1}{E_{\text{tune}}}, \qquad \text{i.e.,} \qquad H(p_{\text{tune}}) \;=\; \gamma E_{\text{tune}} - 1.$$

We now prove the lower bound. Let $\mathcal{T}$ be any deterministic or randomized tree and let $R := \max_{1 \le i \le k} \text{cost}(\mathcal{T}, p_i)/C_i$. By linearity, $\sum_{i=1}^k w_i^* \,\text{cost}(\mathcal{T}, p_i) = \text{cost}(\mathcal{T}, p_{\text{tune}})$. Every deterministic realization has cost at least $C(p_{\text{tune}})$ under p_{tune}, and therefore the same

inequality holds in expectation for a randomized tree, so, by (3.2),

$$\sum_{i=1}^{k} w_i^* \operatorname{cost}(\mathcal{T}, p_i) \;\geq\; C(p_{\text{tune}}) \;\geq\; L(p_{\text{tune}}).$$

By definition of R, $\sum_{i=1}^{k} w_i^* \operatorname{cost}(\mathcal{T}, p_i) \leq R \sum_{i=1}^{k} w_i^* C_i$, so $R \geq L(p_{\text{tune}}) / \sum_{i=1}^{k} w_i^* C_i$.

Using (3.3) together with $H(p_{\text{tune}}) + 1 = \gamma E_{\text{tune}}$,

$$L(p_{\text{tune}}) \;\geq\; H(p_{\text{tune}}) - \log\big(H(p_{\text{tune}}) + 1\big) - 0.45 \;=\; \gamma E_{\text{tune}} - \log(\gamma E_{\text{tune}}) - 1.45.$$

Moreover, for every agent i, $C_i \leq H_i + 1 \leq L_i + \log L_i + 1.45$, so

$$\sum_{i=1}^{k} w_i^* C_i \;\leq\; E_{\text{tune}} + \sum_{i=1}^{k} w_i^* \log L_i + 1.45 \;\leq\; E_{\text{tune}} + \log E_{\text{tune}} + 1.45,$$

again using concavity of the logarithm. Combining the two bounds,

$$R \;\geq\; \frac{\gamma E_{\text{tune}} - \log(\gamma E_{\text{tune}}) - 1.45}{E_{\text{tune}} + \log E_{\text{tune}} + 1.45}.$$

Since $\gamma \geq 1$, this implies $R \geq \gamma - O\big(\gamma(1 + \log E_{\text{tune}})/E_{\text{tune}}\big)$, so $R \geq (1 - o(1))\gamma$ as $E_{\text{tune}} \to \infty$. □

3.3.4 Comparison of the Three Algorithms

The three multiplicative algorithms form a strict hierarchy in terms of the guarantees they provide, with each subsequent algorithm subsuming the previous one.

The nesting relationship $\gamma \leq \lambda$. Since p_{blend} is itself a feasible reference distribution in Definition 3.21, and γ is defined as a minimum over all such distributions,

$$\gamma \;\leq\; \max_{1 \leq i \leq k} \frac{H(p_i, p_{\text{blend}}) + 1}{L_i}.$$

The proof of Theorem 3.18 shows $H(p_i, p_{\text{blend}}) + 1 \leq \lambda L_i$ for every agent i, so $\max_i \big(H(p_i, p_{\text{blend}}) + 1\big)/L_i \leq \lambda$, and hence

$$\gamma \leq \lambda. \tag{3.8}$$

Combined with Proposition 3.19, the three overhead parameters therefore satisfy $\gamma \leq \lambda = O(\log k)$, matching the COMB bound in the worst case and improving on it whenever the agents' cost scales are large or their distributions are well-aligned.

When each algorithm is preferred. COMB (Section 3.3.1) is the simplest: it requires only the optimal trees T_i^* and no distribution information, and achieves $\lceil \log(k+1) \rceil$ overhead unconditionally. BLEND (Section 3.3.2) is preferable when the agents' cost scales are large or when the supports are disjoint, where it is asymptotically tight; it requires only computing λ by a one-dimensional binary search. TUNE (Section 3.3.3) is the right choice when the tightest possible instance-specific bound is needed: by (3.8) it is never worse than BLEND, and it can be strictly better when the agents' supports overlap. The cost is a more expensive optimization (an iterative minimax solve rather than a single scalar equation).

3.4 Robustness to Inaccurate Predictions

The guarantees established so far assume that the supplied access distributions accurately describe future requests. Under prediction error, however, an item assigned an extremely small predicted probability may be placed arbitrarily deep by Lemma 3.7, even if it is frequently accessed in the true workload. We therefore seek robust variants of FUSE, COMB, BLEND, and TUNE that retain their personalized guarantees when predictions are accurate, while ensuring worst-case depth $O(\log n)$ (Definition 3.9) regardless of prediction quality.

We give two robustification methods. The first applies to *any* tree but may lose a factor of two in expected cost. The second applies to algorithms constructed from a reference

distribution, which covers FUSE, BLEND, and TUNE, and preserves their original guarantees up to a vanishing additive term.

3.4.1 A General Method: Depth Capping

Our first transformation fixes a threshold $\delta := \lceil \log n \rceil$, keeps the original tree unchanged through depth δ, and replaces each subtree rooted at depth $\delta + 1$ by a balanced BST on the same items.

Lemma 3.24 (Depth capping)**.** *Let $\widehat{T}$ be the depth-capped tree obtained from a tree T on n items using threshold $\delta := \lceil \log n \rceil$. Then, for every item x,*

$$d_{\widehat{T}}(x) \;\le\; 2\, d_T(x) \qquad \textit{and} \qquad d_{\widehat{T}}(x) \;\le\; 2\lceil \log n \rceil + 1 \;\le\; 2 \log n + 3.$$

Proof. Write $d_j := d_T(x_j)$. If $d_j \le \delta$, item x_j is untouched by the construction, so $d_{\widehat{T}}(x_j) = d_j \le 2d_j$ and $d_{\widehat{T}}(x_j) = d_j \le \delta \le 2\delta + 1$.

If $d_j > \delta$, let v be the ancestor of x_j at depth $\delta + 1$, and let S_v be the subtree rooted at v, containing $m_v \le n$ items. A balanced BST on m_v items has every node at relative depth at most $\lceil \log(m_v + 1) \rceil$, so

$$d_{\widehat{T}}(x_j) \;\le\; \delta + \lceil \log(m_v + 1) \rceil \;\le\; \delta + \lceil \log(n + 1) \rceil \;\le\; 2\delta + 1,$$

using $\lceil \log(n + 1) \rceil \le \delta + 1$: since $n + 1 \le 2n$ for $n \ge 1$, $\log(n + 1) \le 1 + \log n$, and taking ceilings on both sides gives $\lceil \log(n + 1) \rceil \le 1 + \lceil \log n \rceil = \delta + 1$. Since $d_j > \delta$ and both are integers, $d_j \ge \delta + 1$, so $2d_j \ge 2\delta + 2 > 2\delta + 1 \ge d_{\widehat{T}}(x_j)$. Finally $\delta \le \log n + 1$, so $2\delta + 1 \le 2 \log n + 3$. □

Consequently, if T is ρ-multiplicative-optimal, its capped version is 2ρ-multiplicative-optimal. In particular, applying Lemma 3.24 to COMB gives a robust tree satisfying $\mathrm{cost}(\widehat{T}_{\mathrm{comb}}, p_i)/C_i \le 2\lceil \log(k + 1) \rceil$ for every agent i, while every item has depth at most

$2 \log n + 3$.

Depth capping is completely general: it requires neither the agents' distributions nor knowledge of how the original tree was constructed. However, its worst-case analysis incurs a factor-of-two loss even when the predicted distributions are accurate. The method developed next preserves the accurate-prediction guarantee up to a vanishing term while providing the same asymptotic depth bound, but it applies only to trees built from a reference distribution.

3.4.2 A Sharper Method: Floor and Renormalize

For Fuse, Blend, and Tune, the shared tree is obtained by applying the Mehlhorn construction (Lemma 3.7) to a reference distribution q. This additional structure permits a substantially sharper transformation. Before constructing the tree, we give every item a small probability floor and renormalize:

$$\widehat{q}(x) \;:=\; \frac{\max\{q(x), n^{-2}\}}{Z}, \qquad Z \;:=\; \sum_{y \in X} \max\{q(y), n^{-2}\}.$$

The floor n^{-2} is chosen small enough that the resulting distribution does not diverge too much from q, and large enough to guarantee that no item is placed too deep.

Lemma 3.25 (Robust Mehlhorn tree)**.** *Let $\widehat{T}_q$ be the Mehlhorn tree for $\widehat{q}$, and define $\varepsilon_n := \log(1 + 1/n)$. Then, for every item x,*

$$d_{\widehat{T}_q}(x) \;\le\; -\log q(x) + 1 + \varepsilon_n \qquad \textit{and} \qquad d_{\widehat{T}_q}(x) \;\le\; 2 \log n + 2.$$

Proof. Since q sums to 1 and each of the n floor terms contributes at most n^{-2},

$$1 \;=\; \sum_{\ell} q(x_\ell) \;\le\; Z \;\le\; \sum_{\ell} q(x_\ell) + n \cdot n^{-2} \;=\; 1 + \frac{1}{n}.$$

For the first inequality, $\widehat{q}(x_j) \ge q(x_j)/Z \ge q(x_j)/(1 + 1/n)$, so $-\log \widehat{q}(x_j) \le -\log q(x_j) + \log(1 + 1/n)$, and the claim follows from Lemma 3.7 applied to $\widehat{q}$.

For the second inequality, $\widehat{q}(x_j) \geq n^{-2}/Z \geq n^{-2}/(1+1/n) = 1/(n^2+n)$, so

$$-\log \widehat{q}(x_j) \;\leq\; \log n + \log(n+1) \;\leq\; 2\log n + 1,$$

using $\log(n+1) \leq \log n + 1$; adding the $+1$ of Lemma 3.7 gives $d_{\widehat{T}_q}(x_j) \leq 2\log n + 2$. □

The floor is applied only after the reference distribution has been chosen; it therefore does not alter the balancing parameter λ of BLEND or the regret factor γ of TUNE.

Corollary 3.26 (Robust FUSE, BLEND, and TUNE)**.** *Let* $\widehat{\textsc{Fuse}}$, $\widehat{\textsc{Blend}}$, *and* $\widehat{\textsc{Tune}}$ *denote the algorithms obtained by applying Lemma 3.25 to* p_{avg}, p_{blend}, *and* p_{tune}, *respectively. Every resulting tree has maximum depth at most* $2\log n + 2$. *Moreover, for every agent* i,

$$\text{cost}(\widehat{T}_{\text{avg}}, p_i) - C_i \;\leq\; \log k + \log(\log n + 1) + 1.45 + \varepsilon_n,$$

$$\frac{\text{cost}(\widehat{T}_{\text{blend}}, p_i)}{C_i} \;\leq\; \lambda + \varepsilon_n, \qquad \textit{and} \qquad \frac{\text{cost}(\widehat{T}_{\text{tune}}, p_i)}{C_i} \;\leq\; \gamma + \varepsilon_n.$$

Proof. The bounds follow directly by applying floor-and-renormalize to p_{avg}, p_{blend}, and p_{tune}, and then combining the two inequalities of Lemma 3.25 with the analyses of Theorems 3.13, 3.18 and 3.22, respectively: the second inequality gives the depth bound $2\log n + 2$, while the first inequality reproduces each original per-item Mehlhorn bound up to the additive correction ε_n. □

Hence, floor-and-renormalize is the preferred robustification for FUSE, BLEND, and TUNE: it protects every item against arbitrarily inaccurate predictions while preserving the original personalized guarantees up to a small correction. Depth capping remains necessary for COMB, whose construction does not arise from a reference distribution.

Together, Corollary 3.26 and the depth-capped variant of COMB show that every algorithm developed in this chapter can be deployed safely even when the predicted access distributions are unreliable: no item is ever placed deeper than $O(\log n)$, while the personalized fairness

guarantees of Sections 3.2 and 3.3 degrade by at most an additive $\varepsilon_n = \log(1 + 1/n)$, which vanishes as n grows.

Chapter 4

Multi-Agent Framework

Chapter 3 studied one concrete instance of multi-agent fairness: k agents sharing a static binary search tree, each with its own access distribution, where the goal was a tree whose cost, for each agent, stays close to the agent's personalized optimum. Here we generalize that problem. Instead of fixing the shared resource to be a BST, we consider the solution set $\mathbb{S}$ of an arbitrary optimization problem P, shared by k agents $A_1, \ldots, A_k$, each with its own cost function C_x over $\mathbb{S}$. The same tension from Chapter 3 shows up again here: if we just optimize the aggregate cost across agents, some agents can end up much worse off than others, so a fairer approach should instead look for a solution (or a distribution over solutions) where no single agent's cost is disproportionately high.

A few examples:

- *Shared network design.* Several departments in an organization need a spanning tree of an internal network, and each department has its own cost function over the network's links, reflecting how well each link serves that department's needs (for instance, a department running latency-sensitive traffic assigns a high cost to a slow link, even if that link is cheap to build). If we just minimize the aggregate cost across departments, we can easily end up building a network that serves most departments well while leaving one department's critical traffic on links that are particularly poor by its own metric.

- *Fair clustering.* In a multi-agent κ-means or κ-median problem, each agent has its own distance metric over the same points. Minimizing total cost across agents can produce clusters that look great under one agent's metric and bad under another's.

- *Fair matching.* Several agents assign their own weights to the edges of a graph. Rather than maximize total weight, which tends to favor whichever agent's weights happen to be largest, we want a matching that is fair to everyone.

In all of these settings, including the shared-BST problem of Chapter 3, a natural fairness objective is to minimize the *maximum* cost across all agents, so that no single agent is left far behind. The problem is that no deterministic algorithm can achieve this within any bounded factor of what a simple sum-minimizing algorithm already gets. The toy example from Chapter 1 makes this concrete: $k-1$ indivisible cookies are handed out among k agents, and an agent's cost is 1 if it gets no cookie and 0 otherwise. However the cookies are assigned, some agent is always left with none, so a deterministic algorithm's worst-case cost is 1. But a randomized algorithm that hands out the cookies uniformly at random gives every agent an expected cost of only $1/k$, a factor of k better. Randomization is what lets us close this gap: by letting the algorithm output a distribution over solutions instead of single instance, the factor-k gap disappears.

The main contribution of this chapter is a general framework that turns any approximation algorithm for the weighted-sum problem into an approximation algorithm for the min-max problem, keeping essentially the same approximation guarantee and using randomization to close the factor-k gap at the cost of only a small additive term. We give three algorithms that do this, differing in how they search for the agent weighting that exposes the worst-off agent. ROBIN recursively bisects the space of agent weightings. This is enough to give a constructive proof that the factor-k gap can always be closed, but the search is exhaustive: its running time is exponential in the number of agents k, so it is practical only when k is small. SHERWOOD removes that exponential dependence by casting the choice of weights as a repeated game and solving it with the classical Hedge (multiplicative-weights-update)

algorithm from online learning; the tradeoff is a running time that is polynomial, rather than polylogarithmic, in the desired precision. MARIAN improves on both by using tools from convex geometry, namely the ellipsoid method: its running time is polynomial in k *and* logarithmic in the precision.

One practical issue affects all three algorithms as stated: the distributions they output can have exponentially large support. Running a randomized strategy means sampling from this distribution repeatedly, so a large support is a genuine obstacle, and it raises a question of independent interest: how many distinct solutions does a fair outcome need? Section 4.2 shows that a simple, algorithm-agnostic linear program compresses any distribution down to at most $k + 1$ solutions without hurting the approximation guarantee, so support size is not tracked through the rest of the chapter.

The rest of the chapter is organized as follows. Section 4.1 sets up the problems, the zero-sum game and minimax theorem underlying all three algorithms, and the notions of approximation and support we use throughout. Section 4.2 states the main theorems. Sections 4.3 to 4.5 give the three algorithms with full proofs, and Section 4.6 compares the three in detail.

4.1 Preliminaries

Consider a problem P and let $\mathbb{S} = \{S_1, S_2, \ldots\}$ be the set of solutions for this problem. We assume $\mathbb{S}$ is finite. For each $1 \leq x \leq k$, let A_x be an agent, and let C_x be the cost function of A_x on the set of solutions. In other words, for any $s \in \mathbb{S}$, $C_x(s)$ quantifies how undesirable the solution s is for the agent A_x. Throughout the chapter we assume costs are nonnegative and uniformly bounded: there is a constant L_c with $0 \leq C_x(s) \leq L_c$ for every agent A_x and every solution $s \in \mathbb{S}$. All logarithms are base 2.

Randomization was shown above to be necessary; it remains to fix the form it takes. A randomized algorithm is a procedure that flips coins internally and outputs one solution from

$\mathbb{S}$; if we run it again on the same input, it may output a different one. Only the distribution of outputs matters here, not the internal coin flips, so a randomized algorithm can be identified with a probability distribution on $\mathbb{S}$: the coin flips induce a distribution over outputs, and running the algorithm amounts to drawing one sample from that distribution. Searching over randomized procedures is awkward to reason about directly, so we search over distributions on $\mathbb{S}$ instead, and draw an actual solution from the resulting distribution only when one is needed.

Let $\mathcal{F}$ be the set of all probability distributions on $\mathbb{S}$. For a distribution $S \in \mathcal{F}$, the cost of an agent A_x on S is the expected cost of a solution drawn from S, i.e., $C_x(S) = \mathbb{E}_{s\sim S}[C_x(s)]$. Since $\mathbb{S}$ is finite, $\mathcal{F}$ is the probability simplex over $|\mathbb{S}|$ points, which is compact, and $C_x(S)$ is linear (hence continuous) in S; by the extreme value theorem, the minimum in Definition 4.1 below is therefore always attained.

The fairness objective that we study minimizes the worst-case (weighted) cost across all agents.

Definition 4.1 (Wmcm Problem)**.** Given a vector $\mathbf{v}= (v_1, \ldots, v_k) \in \mathbb{R}^k$ of nonnegative weights not exceeding one, the *Weighted Max Cost Minimization* (Wmcm) problem asks for a distribution $S^* \in \mathcal{F}$ that minimizes the maximum weighted cost across all agents:

$$S^* = \operatorname*{argmin}_{S\in\mathcal{F}} \left(\max_{1\le x\le k} (v_x \cdot C_x(S)) \right).$$

We write $C_{OPT} = \min_{S\in\mathcal{F}} \max_{1\le x\le k} (v_x \cdot C_x(S))$ for the optimal value of this problem, i.e. the cost that an algorithm for Wmcm is measured against.

Definition 4.1 only requires that C_x be *some* cost function agent A_x places on solutions in $\mathbb{S}$; it places no further restriction on what that cost measures. This is what lets one problem, Wmcm, capture different notions of fairness. Elsewhere in this chapter $C_x(S)$ is simply the agent's raw cost of S, so Wmcm minimizes the worst raw cost across agents. The shared-BST problem of Chapter 3 instead compares each agent's cost against that agent's *own* best

case: it bounds either the additive overhead $\text{cost}(T, p_x) - C_x^*$ or the multiplicative overhead $\text{cost}(T, p_x)/C_x^*$, where $C_x^* = \min_T \text{cost}(T, p_x)$ is agent A_x's personalized optimal cost. Both notions are instances of WMCM: since C_x^* is a constant fixed by agent A_x alone and does not depend on the shared solution S, setting the cost function to $C_x(S) := \text{cost}(T, p_x) - C_x^*$ (or $\text{cost}(T, p_x)/C_x^*$) still yields a valid, nonnegative, bounded per-agent cost, and minimizing the maximum of this cost function over agents recovers exactly Chapter 3's fairness notion. The framework in this chapter is therefore agnostic to whether C_x measures a raw cost or a cost relative to A_x's own optimum.

For simplicity, without loss of generality, we assume $v_x = 1$ for every agent x: replacing C_x with $v_x \cdot C_x$ absorbs the priority weight into the cost function itself.

The WMCM problem is closely related to the following problem, which is typically easier. Our framework will reduce WMCM to it.

Definition 4.2 (WACM Problem)**.** Given a vector $\mathbf{w} \in \mathbb{R}^k$ of nonnegative weights summing to at most one, the *Weighted Average Cost Minimization* (WACM) problem asks for an instance $s^* \in \mathbb{S}$ that minimizes the weighted total cost:

$$s^* = \operatorname*{argmin}_{s \in \mathbb{S}} \left(\sum_{1 \le x \le k} (w_x \cdot C_x(s)) \right).$$

Since $\mathbb{S}$ is finite, this minimum always exists.

The weights are allowed to sum to less than one for generality, but every invocation made by the framework in this chapter uses a weight vector summing to exactly one, so the reader may safely read $\sum_x w_x = 1$ throughout the analysis.

There is a second way to view C_{OPT}: this alternate view does most of the work in this chapter, since C_{OPT} is the value of a *zero-sum game*, a game between two players whose payoffs are exact opposites, so that whatever one player gains, the other loses. The two players here are an *adversary*, who assigns per-agent weights, and an *optimizer*, who picks a solution to serve them:

- The *optimizer* chooses $Y \in \mathcal{F}$ to minimize cost.
- The *adversary* chooses $\mathbf{u} \in \Delta_k$ to maximize cost, where Δ_k is the set of all ways to divide one unit of "blame" among the k agents: nonnegative weights $u_1, \ldots, u_k$, one per agent, summing to one, with u_x indicating how much weight the adversary places on agent A_x's cost.
- The payoff is $C(Y, \mathbf{u}) = \sum_{x=1}^{k} u_x \cdot C_x(Y)$.

There are two natural ways to play this game, depending on who moves first.

Suppose the optimizer moves first, committing to a solution $Y \in \mathcal{F}$. The adversary, now facing a fixed Y, gets to see all k costs $C_1(Y), \ldots, C_k(Y)$ before choosing $\mathbf{u}$, so it simply puts all its weight on whichever agent currently has the highest cost: this is the best it can do, since $C(Y, \mathbf{u})$ is linear in $\mathbf{u}$ and a linear function over Δ_k is maximized at a vertex. The resulting payoff is $\max_{\mathbf{u} \in \Delta_k} C(Y, \mathbf{u}) = \max_{1 \le x \le k} C_x(Y)$. Anticipating this, the optimizer's best choice is a Y that makes this worst-case cost as small as possible, i.e. that minimizes $\max_{1 \le x \le k} C_x(Y)$, which is exactly the WMCM objective we are trying to solve.

Now suppose instead the adversary moves first, committing to a weight vector $\mathbf{u} \in \Delta_k$. The optimizer, facing this fixed $\mathbf{u}$, simply solves WACM at weights $\mathbf{u}$: minimizing $C(Y, \mathbf{u}) = \sum_x u_x C_x(Y)$ over Y is exactly the WACM objective. Anticipating this, the adversary's problem becomes choosing a weight vector $\mathbf{u}$ that makes the resulting WACM value, $\min_{Y \in \mathcal{F}} C(Y, \mathbf{u})$, as large as possible; that is, finding the weights that are hardest for WACM to satisfy well.

These two games need not have the same value in general: moving first can only put a player at a disadvantage, never an advantage, so

$$\max_{\mathbf{u} \in \Delta_k} \min_{Y \in \mathcal{F}} C(Y, \mathbf{u}) \le \min_{Y \in \mathcal{F}} \max_{\mathbf{u} \in \Delta_k} C(Y, \mathbf{u})$$

always holds, but equality is not automatic. Whether it holds here, i.e. whether it matters who commits first, is the question the minimax theorem below answers. This matters for us because the second game is the one our algorithms can actually play: evaluating

$\min_{Y \in \mathcal{F}} C(Y, \mathbf{u})$ at any single weight vector $\mathbf{u}$ costs exactly one WACM solve, so computing $\max_{\mathbf{u} \in \Delta_k} \min_{Y \in \mathcal{F}} C(Y, \mathbf{u})$ reduces to searching over weight vectors $\mathbf{u}$ and calling WACM at each one, which is exactly what ROBIN, SHERWOOD, and MARIAN do below, each with a different search strategy.

Theorem 4.3 (Wald's Minimax Theorem [Wald, 1945])**.** *Let $\mathcal{F}$ be a set of solutions and $U \subseteq \mathbb{R}^k$ a compact convex set of weight vectors. If $C : \mathcal{F} \times U \to \mathbb{R}$ is linear in Y for every fixed $\mathbf{u} \in U$, and linear in $\mathbf{u}$ for every fixed $Y \in \mathcal{F}$, then*

$$\min_{Y \in \mathcal{F}} \max_{\mathbf{u} \in U} C(Y, \mathbf{u}) \;=\; \max_{\mathbf{u} \in U} \min_{Y \in \mathcal{F}} C(Y, \mathbf{u}).$$

Corollary 4.4. *Applying Theorem 4.3 with $U = \Delta_k$,*

$$C_{OPT} \;=\; \min_{Y \in \mathcal{F}} \max_{\mathbf{u} \in \Delta_k} C(Y, \mathbf{u}) \;=\; \max_{\mathbf{u} \in \Delta_k} \min_{Y \in \mathcal{F}} C(Y, \mathbf{u}).$$

Each of the three algorithms below is a different strategy for evaluating one side of this identity using only calls to the other.

With both problems defined, we now fix how we will compare algorithms along two axes: how close they get to the optimum, and how many distinct solutions their output distribution needs.

Definition 4.5 (Approximation)**.** For a problem P, an algorithm A is said to be a (δ, β)-approximation of P if

$$C_A \leq \delta \cdot C^* + \beta,$$

where C_A is the cost of what A returns, and C^* is the cost of the optimal solution of P.

This captures both multiplicative approximation (the factor δ) and additive approximation (the term β) at once; an exact algorithm is the special case of a $(1, 0)$-approximation. The two-parameter form is what lets the framework accept such a wide range of existing algorithms as the underlying solver.

The other axis we care about is how many solutions an algorithm's output actually spans, which we capture with the following notion.

Definition 4.6 (Support)**.** For a distribution $S \in \mathcal{F}$, the *support* of S is $\operatorname{supp}(S) = \{s \in \mathbb{S} : \Pr[S = s] > 0\}$, and $|\operatorname{supp}(S)|$ is its *support size*. A randomized algorithm A that returns a distribution in $\mathcal{F}$ on every instance is said to have support size (at most) N if $|\operatorname{supp}(A(\cdot))| \leq N$ on every instance.

Support size matters practically as well as theoretically: if an algorithm's output has support size N, running the randomized strategy requires being able to produce any of these N solutions on demand.

4.2 Main Results

Let Alg be an algorithm that returns a (δ, β)-approximation of the Wacm problem in time T_{Alg}, and suppose Alg also reports the cost of each agent on its output, so those costs can be checked cheaply. Recall from Section 4.1 that L_c is the uniform upper bound on all agents' costs, $0 \leq C_x(s) \leq L_c$. We give three algorithms for Wmcm built on top of Alg, each with a different strength.

All three algorithms are *deterministic* in the sense that the distribution they return is constructed deterministically from the input; the randomness lives only in the final step of drawing a solution from that distribution.

One bookkeeping concern can be disposed of first: none of these algorithms needs to work to keep its output distribution small. For any distribution $S \in \mathcal{F}$ with finite support size N, enumerate its support as $Y^{(1)}, \ldots, Y^{(N)}$ with probabilities $q_1, \ldots, q_N$, and solve the auxiliary linear program

$$\min v \quad \text{s.t.} \quad \sum_{t=1}^{N} p_t \, C_x(Y^{(t)}) \leq v \;\; \forall x, \qquad \sum_{t=1}^{N} p_t = 1, \;\; p_t \geq 0.$$

The q_t themselves are a feasible point with objective value $\max_x C_x(S)$, so the LP's optimum v^* satisfies $v^* \leq \max_x C_x(S)$. The LP has N variables but only $k+1$ structural constraints, so it has an optimal solution p^* that is a basic feasible solution, with at most $k+1$ strictly positive entries; drawing $Y^{(t)}$ with probability p_t^* gives a distribution $S' \in \mathcal{F}$ with support size at most $k+1$ and $\max_x C_x(S') = v^* \leq \max_x C_x(S)$, computable in $O(k^2 N)$ time, via $O(N-(k+1))$ pivot steps, each eliminating one nonzero variable in $O(k^2)$ work. Therefore, for the rest of this chapter, the support size of an algorithm's output needs no further attention: whatever pool of solutions an algorithm queries along the way, this single LP compresses it to at most $k+1$ solutions.

Theorem 4.7 (Robin)**.** *For any $\epsilon > 0$, there is a deterministic algorithm Robin (**R**ecursive **O**racle-**B**ased **I**nterval **N**arrowing) that returns a $(\delta, \beta+\epsilon)$-approximation of the Wmcm problem, in time*

$$O\big(T_{\textsc{Alg}} \cdot \log^{k-1}(4kL_c/\epsilon)\big)\,.$$

Note the running time is exponential in k; that is the price of Robin's exhaustive recursive search over weight assignments. The next algorithm eliminates that exponential dependence, but pays for it with a worse dependence on the precision ϵ.

Theorem 4.8 (Sherwood)**.** *For any $\epsilon > 0$, there is a deterministic algorithm Sherwood that returns a $(\delta, \beta+\epsilon)$-approximation of the Wmcm problem, in time*

$$O\big(T_{\textsc{Alg}} \cdot L_c^2 \ln(k)/\epsilon^2\big)\,.$$

Sherwood borrows from online learning: it treats the choice of weights as a zero-sum game played repeatedly, and solves it with the classical *Hedge* algorithm [Freund and Schapire, 1999, Arora et al., 2012], using Alg as the best-response oracle at each round. Hedge is the standard tool for computing the value of such a game.

The two algorithms carry the same approximation guarantee (the multiplicative factor δ is unchanged, and the additive term grows only by the controllable ϵ), but they scale very

differently with the two parameters that matter:

- ROBIN's recursive binary search is polylogarithmic in $1/\epsilon$ and L_c, but exponential in k, so it makes sense when k is small and precision matters.
- SHERWOOD's Hedge-based search is logarithmic in k, but only polynomial ($1/\epsilon^2$) in the precision, so it makes sense when k is large and moderate precision is fine.

Neither is satisfactory when both the agent count and the required precision are large. The third algorithm addresses this, at the cost of a more involved construction based on convex geometry rather than search or online learning.

Theorem 4.9 (MARIAN)**.** *For any $\epsilon > 0$, the algorithm MARIAN (**M**in-max **A**pproximation via **R**egion-**I**ntersection **A**nd **N**arrowing) returns a $(\delta, \beta + \epsilon)$-approximation of the WMCM problem, in time*

$$O\left(T_{ALG} \cdot k^2 \log \frac{kL_c}{\epsilon}\right).$$

MARIAN is built on ellipsoid methods, a classical family of algorithms that shrink the search space of a problem step by step until only near-optimal points are left [Khachiyan, 1979, Grötschel et al., 1988]. The special case where ALG solves WACM exactly is not new: it is essentially the maxmin-fair combinatorial search algorithm of García-Soriano and Bonchi [García-Soriano and Bonchi, 2021] (Section 2.5.2). Our contribution here is showing that the same idea still works when ALG is only an approximate (δ, β)-oracle, which is what lets us plug it into the rest of the framework.

MARIAN matches or beats both ROBIN and SHERWOOD on running time (see Section 4.6 for the details), so in principle it is always the algorithm to use. ROBIN and SHERWOOD are, however, much simpler: a recursive binary search in one case and a handful of multiplicative-weight updates in the other, against the ellipsoid machinery behind MARIAN. Table 4.1 lays out the running time each algorithm guarantees.

The full algorithm descriptions and proofs for the following theorems are in Sections 4.3 to 4.5.

Table 4.1: Running time for the three algorithms.

	Running time
ROBIN	$O\big(T_{\text{ALG}} \cdot \log^{k-1}(4kL_c/\epsilon)\big)$
SHERWOOD	$O(T_{\text{ALG}} \cdot L_c^2 \ln(k)/\epsilon^2)$
MARIAN	$O(T_{\text{ALG}} \cdot k^2 \log(kL_c/\epsilon))$

4.3 The ROBIN Algorithm

We now present the first algorithm that realizes Theorem 4.7. The acronym stands for **R**ecursive **O**racle-**B**ased **I**nterval **N**arrowing. The name also evokes Robin Hood, as the algorithm redistributes cost from the agents who have too little of the optimizer's attention to those who have too much.

4.3.1 The Algorithm

We start with the algorithm for $k = 2$ and $k = 3$ agents, where the underlying idea is easiest to see, and then give the formalized algorithm for general k.

That idea is to locate, by binary search, a weighting at which the agents' costs are balanced. Since the balance point cannot be hit exactly, the algorithm randomizes between the two solutions bracketing it, so that the expected costs equalize exactly.

Two agents ($k = 2$). With two agents, the algorithm searches for weights $\mathbf{w} = (1 - \alpha, \alpha)$, where α acts as an artificial priority assigned to agent A_2, and $1 - \alpha$ to agent A_1. The WACM solution under these weights is then used as a WMCM solution. If A_2's cost exceeds A_1's under a given weight assignment, it makes sense to increase α so the oracle prioritizes A_2 more.

More specifically, ROBIN starts at $\alpha_\ell = 0$, running WACM for $\mathbf{w} = (1, 0)$ and calling the output X_ℓ. If $C_2(X_\ell) \leq C_1(X_\ell)$, the algorithm returns X_ℓ: agent A_2 is already no worse off than A_1 even with zero priority, so no redistribution is needed. Symmetrically, WACM is executed for $\mathbf{w} = (0, 1)$ to produce X_r with $\alpha_r = 1$. If $C_1(X_r) \leq C_2(X_r)$, the algorithm

returns X_r, as A_2 remains the worse-off agent even with maximum priority.

If neither boundary resolves the problem, the algorithm holds two solutions, X_ℓ and X_r, satisfying the "base inequalities": $C_2(X_\ell) > C_1(X_\ell)$ and $C_1(X_r) > C_2(X_r)$. Because the relative ordering of costs *flips* between the two endpoints, ROBIN performs a binary search. At each step, it evaluates $\mathbf{w} = (1 - mid, mid)$ where $mid = (\alpha_\ell + \alpha_r)/2$, and replaces whichever boundary shares the same cost ordering as the new solution, preserving the base inequalities. Once the bracket is narrow enough, the algorithm outputs a randomized mixture: X_ℓ with probability p and X_r with probability $1 - p$, choosing p to equalize the expected costs perfectly.

Three agents ($k = 3$). For three agents, ROBIN recursively searches for weights $\mathbf{w} = (w_1, w_2, w_3)$ summing to one. First, it fixes $w_3 = 0$ and runs the two-agent algorithm on A_1 and A_2 to obtain X_ℓ. Next, it fixes $w_3 = 1$ and runs it again to obtain X_r. The outer level then binary searches over w_3 using the same flip-and-narrow strategy, balancing A_3 against $C_{[2]} = \max(C_1, C_2)$. This recursive structure generalizes naturally to any number of agents: level s balances A_s against the $s - 1$ agents that level $s - 1$ has already balanced.

We now formalize this for general k. We begin with a definition to simplify the presentation.

Definition 4.10. For an integer $1 \leq x \leq k$ and a distribution $X \in \mathcal{F}$, let $C_{[x]}(X) = \max(C_1(X), \ldots, C_x(X))$.

Recall from Section 4.1 that each agent's priority weight v_x has already been absorbed into C_x, so $C_{[k]}$ above is exactly the WMCM objective.

The ROBIN algorithm is parameterized by a positive real number ϵ (controlling precision) and an integer $1 \leq s \leq k$ (defining the current agent being balanced). It receives inputs ϵ, s, and a weight vector $\mathbf{w} = (w_1, \ldots, w_k)$. It computes optimal weights $\mathbf{w}^* = (w_1^*, \ldots, w_s^*, w_{s+1}, \ldots, w_k)$ summing to one, and returns a distribution $X_{s,\mathbf{w}} \in \mathcal{F}$. The top-level call ROBIN$(\epsilon, k, \mathbf{0})$ solves the general WMCM problem.

One subtlety in the pseudocode below: at level s, only the coordinates $w_{s+1}, \ldots, w_k$

actually mean anything, since these are the priorities fixed by higher levels. The coordinates $w_1, \ldots, w_s$ are just placeholders that get filled in as the recursion goes deeper, and whatever mass is still unassigned once the base case is reached is assigned entirely to agent A_1.

Accordingly, the base case ($s = 1$) simply returns the output of ALG on the WACM instance with weights $\mathbf{w}' = (1 - \sum_{j=2}^{k} w_j, w_2, \ldots, w_k)$. For $s > 1$, the algorithm proceeds in three phases.

Phase 1: Initialization. The algorithm initializes a search interval $[0, 1-(w_{s+1}+\cdots+w_k)]$ for the optimal value of w_s^*, and computes boundary solutions:

1. Call ROBIN$(\epsilon, s-1, \mathbf{w}_\ell)$ with $\mathbf{w}_\ell = (0, \ldots, 0, w_{s+1}, \ldots, w_k)$ to obtain X_ℓ, and set $\alpha_\ell = 0$. If $C_s(X_\ell) \leq C_{[s-1]}(X_\ell)$, return X_ℓ (agent A_s's cost is already dominated).

2. Call ROBIN$(\epsilon, s-1, \mathbf{w}_r)$ with $\mathbf{w}_r = (0, \ldots, 0, 1-(w_{s+1}+\cdots+w_k), w_{s+1}, \ldots, w_k)$ to obtain X_r, and set $\alpha_r = 1-(w_{s+1}+\cdots+w_k)$. If $C_{[s-1]}(X_r) \leq C_s(X_r)$, return X_r (agent A_s dominates even with maximum priority).

If neither condition triggers, the algorithm holds the base inequalities $C_s(X_\ell) > C_{[s-1]}(X_\ell)$ and $C_{[s-1]}(X_r) > C_s(X_r)$, and enters the update phase.

Phase 2: Binary Search Update. The algorithm repeatedly bisects the search interval $[\alpha_\ell, \alpha_r]$. At each step:

- Compute $mid = (\alpha_\ell + \alpha_r)/2$ and let

$$\mathbf{w}_{mid} = (0, \ldots, 0, mid, w_{s+1}, \ldots, w_k).$$

  Call ROBIN$(\epsilon, s-1, \mathbf{w}_{mid})$ to obtain X_{mid}.

- If $C_s(X_{mid}) \leq C_{[s-1]}(X_{mid})$, update the right boundary: $X_r \leftarrow X_{mid}$, $\alpha_r \leftarrow mid$.

- Otherwise, update the left boundary: $X_\ell \leftarrow X_{mid}$, $\alpha_\ell \leftarrow mid$.

Stopping Criterion. Let $\text{slack}_\ell = C_s(X_\ell) - C_{[s-1]}(X_\ell) \geq 0$ and $\text{slack}_r = C_{[s-1]}(X_r) - C_s(X_r) \geq 0$. We quantify how far the algorithm is from perfect equalization by defining

$$\text{error} = (\alpha_r - \alpha_\ell) \cdot \frac{\text{slack}_\ell \cdot \text{slack}_r}{\text{slack}_\ell + \text{slack}_r}.$$

This is the algebraically equivalent, division-safe form of $(\alpha_r - \alpha_\ell)/(1/\text{slack}_\ell + 1/\text{slack}_r)$: the numerator $(\alpha_r - \alpha_\ell)$ measures how much of the search interval remains to be closed, while the denominator grows large exactly when either endpoint is already nearly balanced, so error shrinks as either the interval narrows or the current solutions approach the balance point. Whenever a slack reaches 0, error is exactly 0, which immediately satisfies the termination criterion below; the algorithm therefore stops as soon as one slack hits 0, so the division above is only ever evaluated while both slacks are strictly positive. The algorithm terminates when error $\leq \epsilon/k$. Because both slacks are bounded by L_c and the interval halves every iteration, termination is guaranteed within $\lceil \log(kL_c/(2\epsilon)) \rceil$ steps.

Phase 3: Output Construction. The algorithm computes a mixing probability

$$p = \frac{\text{slack}_r}{\text{slack}_\ell + \text{slack}_r} \in [0, 1],$$

and returns a random variable X that draws from X_ℓ with probability p, and from X_r with probability $1 - p$. This value of p is chosen to ensure

$$p \cdot C_s(X_\ell) + (1-p) \cdot C_s(X_r) = p \cdot C_{[s-1]}(X_\ell) + (1-p) \cdot C_{[s-1]}(X_r), \tag{4.1}$$

which simplifies to $p \cdot \text{slack}_\ell = (1-p) \cdot \text{slack}_r$, satisfying our formulation for p.

The complete pseudocode is given in Algorithm 2.

Algorithm 2 ROBIN$(\epsilon, s, \mathbf{w})$

Require: Precision $\epsilon > 0$, agent index $1 \le s \le k$, weight vector $\mathbf{w} = (w_1, \ldots, w_k)$
Ensure: A distribution $X \in \mathcal{F}$ approximating the optimal WMCM solution

if $s = 1$ **then**
  return ALG$(\mathbf{w}')$ where $\mathbf{w}' = (1 - \sum_{j=2}^{k} w_j, w_2, \ldots, w_k)$
end if
▷ Phase 1: Initialization
$\mathbf{w}_\ell \leftarrow (0, \ldots, 0, w_{s+1}, \ldots, w_k)$
$X_\ell \leftarrow \text{ROBIN}(\epsilon, s-1, \mathbf{w}_\ell), \quad \alpha_\ell \leftarrow 0$
if $C_s(X_\ell) \le C_{[s-1]}(X_\ell)$ **then**
  return X_ℓ
end if
$\mathbf{w}_r \leftarrow (0, \ldots, 0, 1 - \sum_{j=s+1}^{k} w_j, w_{s+1}, \ldots, w_k)$
$X_r \leftarrow \text{ROBIN}(\epsilon, s-1, \mathbf{w}_r), \quad \alpha_r \leftarrow 1 - \sum_{j=s+1}^{k} w_j$
if $C_{[s-1]}(X_r) \le C_s(X_r)$ **then**
  return X_r
end if
▷ Phase 2: Binary Search Update
repeat
  $mid \leftarrow \frac{\alpha_\ell + \alpha_r}{2}$
  $\mathbf{w}_{mid} \leftarrow (0, \ldots, 0, mid, w_{s+1}, \ldots, w_k)$
  $X_{mid} \leftarrow \text{ROBIN}(\epsilon, s-1, \mathbf{w}_{mid})$
  if $C_s(X_{mid}) \le C_{[s-1]}(X_{mid})$ **then**
    $X_r \leftarrow X_{mid}, \quad \alpha_r \leftarrow mid$
  else
    $X_\ell \leftarrow X_{mid}, \quad \alpha_\ell \leftarrow mid$
  end if
  $slack_\ell \leftarrow C_s(X_\ell) - C_{[s-1]}(X_\ell)$ ▷ > 0 by the update rule
  $slack_r \leftarrow C_{[s-1]}(X_r) - C_s(X_r)$ ▷ ≥ 0 by the update rule
  $error \leftarrow (\alpha_r - \alpha_\ell) \cdot \dfrac{slack_\ell \cdot slack_r}{slack_\ell + slack_r}$
until $error \le \epsilon / k$
▷ Phase 3: Output Construction
$p \leftarrow \frac{slack_r}{slack_\ell + slack_r}$
return random variable X that draws from X_ℓ with probability p, and from X_r with probability $1 - p$

4.3.2 Analysis

The analysis proceeds in three parts: notation and setup, five supporting lemmas, and an induction on the recursion depth.

Notation and Setup

Definition 4.11. For any $Y \in \mathcal{F}$ and a weight vector $\mathbf{u} = (u_1, \ldots, u_k)$ summing to one, the weighted-sum cost is $C(Y, \mathbf{u}) = \sum_{x=1}^{k} u_x \cdot C_x(Y)$. The min-max cost for the first s agents is:

$$C_{MM}^{(s)}(Y, \mathbf{u}) = (1 - u_s^+) \cdot C_{[s]}(Y) + \sum_{x>s} u_x \cdot C_x(Y), \tag{4.2}$$

where $u_s^+ = \sum_{x>s} u_x$.

The family $C_{MM}^{(s)}$ interpolates between the two objectives at stake. ALG minimizes a weighted *sum* of agent costs, whereas the objective we want, $C_{[k]}$, is a *max*; (4.2) takes the max over the first s agents and the weighted sum over the rest, so that $C_{MM}^{(k)}(Y, \mathbf{u}) = C_{[k]}(Y)$ is the WMCM objective while $C_{MM}^{(1)}(Y, \mathbf{u}) = C(Y, \mathbf{u})$ is the WACM objective. Each recursive level of ROBIN moves one agent from the sum side to the max side. A second property, used just as often below, is that $C_{MM}^{(s)}(Y, \mathbf{u})$ depends on $\mathbf{u}$ only through the coordinates $u_{s+1}, \ldots, u_k$.

Remark 4.12. We assume that s is fixed for the rest of this section, or equivalently, we are analyzing a fixed layer of the algorithm. For any weight vector $\mathbf{u}$ summing to one, let $C(\mathbf{u}) = \inf_{Y \in \mathcal{F}} C(Y, \mathbf{u})$ denote the minimum weighted-sum cost, and $C_{MM}^{(s)}(\mathbf{u}) = \inf_{Y \in \mathcal{F}} C_{MM}^{(s)}(Y, \mathbf{u})$ the minimum min-max cost.

To establish the "worst-case weights" at recursion level s, we restrict which coordinates may vary: agents $1, \ldots, s$ are free, while agents $s+1, \ldots, k$ are pinned to the values passed down by higher levels of ROBIN. Let $S_{s,\mathbf{w}} = \{\mathbf{u}' \mid \forall x : u'_x \geq 0,\ \forall x > s : u'_x = w_x,\ \sum_x u'_x = 1\}$ denote this set of feasible weight vectors. Within it, define the optimal (worst-case) weight vector

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{u} \in S_{s,\mathbf{w}}} C(\mathbf{u}),$$

and similarly $\mathbf{w}_\ell^* = \operatorname{argmax}_{\mathbf{u} \in S_{s-1,\mathbf{w}_\ell}} C(\mathbf{u})$ and $\mathbf{w}_r^* = \operatorname{argmax}_{\mathbf{u} \in S_{s-1,\mathbf{w}_r}} C(\mathbf{u})$, where $\mathbf{w}_\ell$ and $\mathbf{w}_r$ denote the weight vectors of whichever calls produced the current X_ℓ and X_r (these get

overwritten during Phase 2, so $\mathbf{w}_\ell, \mathbf{w}_r$ always refer to the most recent ones). The lemmas below describe relationships between these three vectors, $\mathbf{w}^*$, $\mathbf{w}_\ell^*$, and $\mathbf{w}_r^*$.

Supporting Lemmas

Five lemmas precede the induction. The first establishes that $\mathbf{w}^*$, $\mathbf{w}_\ell^*$, and $\mathbf{w}_r^*$ all exist.

Lemma 4.13. *For every level $s' \in \{1, \ldots, k\}$ and every feasible weight vector $\mathbf{w}'$, the argmax of $C(\mathbf{u})$ is well defined in $S_{s',\mathbf{w}'}$.*

Proof. For a fixed Y, $C(Y, \mathbf{u})$ is linear in $\mathbf{u}$ with coefficients bounded by L_c, making it L_c-Lipschitz, and an infimum of Lipschitz functions is Lipschitz, so $C(\mathbf{u})$ is continuous. $S_{s',\mathbf{w}'}$ is compact, so the Extreme Value Theorem gives that the maximum is attained. □

Remark 4.14. Instantiating at $(s', \mathbf{w}') = (s, \mathbf{w})$ gives that $\mathbf{w}^*$ exists, and at $(s-1, \mathbf{w}_\ell)$ and $(s-1, \mathbf{w}_r)$ gives that $\mathbf{w}_\ell^*$ and $\mathbf{w}_r^*$ exist.

The next lemma is where Wald's minimax theorem enters, applied over the restricted weight set $S_{s,\mathbf{w}}$ rather than the full simplex Δ_k: an adversary choosing weights and an optimizer choosing a solution reach the same value regardless of who commits first. It lets $C_{MM}^{(s)}(\mathbf{w}^*)$ be exchanged for $C(\mathbf{w}^*)$, the quantity the recursion actually controls.

Lemma 4.15. *For every level $s' \in \{1, \ldots, k\}$ and every feasible weight vector $\mathbf{w}'$, writing $\mathbf{w}'^* = \operatorname{argmax}_{\mathbf{u} \in S_{s',\mathbf{w}'}} C(\mathbf{u})$, $C_{MM}^{(s')}(\mathbf{w}'^*) = C(\mathbf{w}'^*)$.*

Proof. $C(Y, \mathbf{u})$ is linear in $\mathbf{u}$ for fixed Y, and, identifying each $Y \in \mathcal{F}$ with its probability vector over $\mathbb{S}$, linear in Y for fixed $\mathbf{u}$: $C(Y, \mathbf{u}) = \sum_x u_x \sum_{s \in \mathbb{S}} Y(s) C_x(s)$ is a linear functional of Y's coordinates. As $S_{s',\mathbf{w}'}$ is compact, Theorem 4.3 permits swapping the optimization order:

$$\max_{\mathbf{u} \in S_{s',\mathbf{w}'}} \inf_Y C(Y, \mathbf{u}) = \inf_Y \max_{\mathbf{u} \in S_{s',\mathbf{w}'}} C(Y, \mathbf{u}).$$

The left side evaluates to $C(\mathbf{w}'^*)$ by definition. For the right side, fixing Y pins coordinates u_x for $x > s'$ to w'_x. The linear objective on the remaining simplex is maximized by placing all

unassigned mass on the highest-cost agent among $A_1, \ldots, A_{s'}$, yielding $\max_{\mathbf{u} \in S_{s',\mathbf{w}'}} C(Y, \mathbf{u}) = C^{(s')}_{MM}(Y, \mathbf{w}')$. Taking $\inf_Y$ results in $C^{(s')}_{MM}(\mathbf{w}') = C^{(s')}_{MM}(\mathbf{w}'^*)$. □

Remark 4.16. Instantiating at $(s', \mathbf{w}') = (s, \mathbf{w})$ gives $C^{(s)}_{MM}(\mathbf{w}^*) = C(\mathbf{w}^*)$, and at $(s-1, \mathbf{w}_\ell)$ and $(s-1, \mathbf{w}_r)$ gives the same identity for $\mathbf{w}^*_\ell$ and $\mathbf{w}^*_r$.

Lemma 4.17. $C^{(s-1)}_{MM}(\mathbf{w}^*_\ell) \le C^{(s)}_{MM}(\mathbf{w}^*)$, *and similarly for* $\mathbf{w}^*_r$.

Proof. Because $S_{s-1,\mathbf{w}_\ell} \subseteq S_{s,\mathbf{w}}$, it holds that $C(\mathbf{w}^*_\ell) = \max_{S_{s-1,\mathbf{w}_\ell}} C(\mathbf{u}) \le \max_{S_{s,\mathbf{w}}} C(\mathbf{u}) = C(\mathbf{w}^*)$. Apply Lemma 4.15 at $(s-1, \mathbf{w}_\ell)$ and at $(s, \mathbf{w})$. □

Lemma 4.17 is what justifies the early exit cases in the algorithm: going from s agents down to $s-1$ can only lower the worst-case cost, never raise it.

Lemma 4.17 covers the case where ROBIN exits during Phase 1. When the binary search runs and returns a mixture, two further facts are needed. The first is that the mixture's cost is the p-weighted average of the two boundary costs.

Lemma 4.18. $C^{(s)}_{MM}(X, \mathbf{w}^*) = p \cdot C^{(s-1)}_{MM}(X_\ell, \mathbf{w}^*) + (1-p) \cdot C^{(s-1)}_{MM}(X_r, \mathbf{w}^*)$.

Proof. Let $M = p \cdot C_{[s-1]}(X_\ell) + (1-p) \cdot C_{[s-1]}(X_r)$. The claim is not immediate, since $C_{[s]}$ and $C_{[s-1]}$ are maxima rather than linear functions of the mixture. Each individual cost C_x *is* linear over mixtures, however, so $C_x(X) = p \cdot C_x(X_\ell) + (1-p) \cdot C_x(X_r)$ for every x. For agent s this equals M by the choice of p in (4.1). For any $x < s$ we instead have $C_x(X_\ell) \le C_{[s-1]}(X_\ell)$ and $C_x(X_r) \le C_{[s-1]}(X_r)$, so $C_x(X) \le M$ as well, and taking the max over these x gives $C_{[s-1]}(X) \le M$. So $C_{[s]}(X) = \max(C_{[s-1]}(X), C_s(X)) = M$, since the second term already equals M.

Expanding both sides of the lemma with (4.2) completes the proof. On the left, $C^{(s)}_{MM}(X, \mathbf{w}^*) = (1-u^+)M + \sum_{x>s} w^*_x C_x(X)$ where $u^+ = \sum_{x>s} w^*_x$. On the right, applying (4.2) to each of X_ℓ and X_r at level $s-1$ and combining with weights $p, 1-p$, Writing

$\Sigma^* = \sum_{x \geq s} w_x^*$ for brevity,

$$
\begin{aligned}
p \cdot C_{MM}^{(s-1)}(X_\ell, \mathbf{w}^*) + (1-p) \cdot C_{MM}^{(s-1)}(X_r, \mathbf{w}^*) \\
&= p\big(1-\Sigma^*\big) C_{[s-1]}(X_\ell) + p \textstyle\sum_{x \geq s} w_x^* C_x(X_\ell) \\
&\quad + (1-p)\big(1-\Sigma^*\big) C_{[s-1]}(X_r) + (1-p) \textstyle\sum_{x \geq s} w_x^* C_x(X_r) \\
&= \big(1-\Sigma^*\big)\big(p\, C_{[s-1]}(X_\ell) + (1-p)\, C_{[s-1]}(X_r)\big) \\
&\quad + \textstyle\sum_{x \geq s} w_x^* \big(p\, C_x(X_\ell) + (1-p)\, C_x(X_r)\big) \\
&= \big(1-\Sigma^*\big) \cdot M + \textstyle\sum_{x \geq s} w_x^*\, C_x(X) \\
&= \big(1 - \textstyle\sum_{x > s} w_x^*\big) \cdot M - w_s^* M + w_s^*\, C_s(X) + \textstyle\sum_{x > s} w_x^*\, C_x(X),
\end{aligned}
$$

which, since $C_s(X) = M$ as established above, collapses the $-w_s^* M + w_s^*\, C_s(X)$ term to 0, leaving exactly $(1-u^+)M + \sum_{x>s} w_x^* C_x(X)$, matching the left side. □

The recursive call that produced X_ℓ was solved for the boundary weight vector $\mathbf{w}_\ell^*$, not for the worst case $\mathbf{w}^*$ at this level. The gap between the two is exactly the slack from the binary search.

Lemma 4.19. *$C_{MM}^{(s-1)}(X_\ell, \mathbf{w}^*) = C_{MM}^{(s-1)}(X_\ell, \mathbf{w}_\ell^*) + (w_s^* - \alpha_\ell) \cdot slack_\ell$, and $C_{MM}^{(s-1)}(X_r, \mathbf{w}^*) = C_{MM}^{(s-1)}(X_r, \mathbf{w}_r^*) + (\alpha_r - w_s^*) \cdot slack_r$.*

Proof. $\mathbf{w}^*$ and $\mathbf{w}_\ell^*$ agree on all coordinates $> s$, and $C_{MM}^{(s-1)}(Y, \mathbf{u})$ depends on $\mathbf{u}$ only through coordinates $\geq s$, so $\mathbf{w}^*$ and $\mathbf{w}_\ell^*$ can only differ at coordinate s: it is w_s^* under $\mathbf{w}^*$, versus α_ℓ under $\mathbf{w}_\ell^*$. Shifting that one coordinate from α_ℓ to w_s^* moves exactly that much mass off $C_{[s-1]}$ and onto C_s:

$$
C_{MM}^{(s-1)}(X_\ell, \mathbf{w}^*) - C_{MM}^{(s-1)}(X_\ell, \mathbf{w}_\ell^*) = (w_s^* - \alpha_\ell)\,\big(C_s(X_\ell) - C_{[s-1]}(X_\ell)\big) = (w_s^* - \alpha_\ell) \cdot slack_\ell.
$$

The same computation for X_r gives $(w_s^* - \alpha_r)(C_s(X_r) - C_{[s-1]}(X_r))$, which is $(\alpha_r - w_s^*) \cdot slack_r$ once we account for $slack_r = C_{[s-1]}(X_r) - C_s(X_r)$ having the opposite sign convention from

$slack_\ell$. □

Main Inductive Argument

Lemma 4.20. $C^{(s)}_{MM}(X, \mathbf{w}^*) \leq \delta \cdot C^{(s)}_{MM}(\mathbf{w}^*) + \beta + \frac{s}{k}\epsilon$.

Proof. The proof proceeds by induction on the depth s.

Base case ($s = 1$). Here, X is the unmodified output of ALG using

$$\mathbf{w}' = \Big(1 - \sum_{j \geq 2} w_j,\ w_2, \ldots, w_k\Big).$$

Because $S_{1,\mathbf{w}}$ has no free coordinates, $\mathbf{w}^* = \mathbf{w}'$. Given $C^{(1)}_{MM}(Y, \mathbf{u}) = C(Y, \mathbf{u})$, the (δ, β)-approximation guarantee of ALG directly yields:

$$C^{(1)}_{MM}(X, \mathbf{w}^*) = C(X, \mathbf{w}^*) \leq \delta \cdot C(\mathbf{w}^*) + \beta = \delta \cdot C^{(1)}_{MM}(\mathbf{w}^*) + \beta.$$

Inductive step. Assume the bound holds at depth $s - 1$. At level s, ROBIN either triggers an early initialization exit or completes the binary search mixture.

Case 1: initialization exit. Suppose $X = X_\ell$ (the case $X = X_r$ is symmetric). The exit condition $C_s(X_\ell) \leq C_{[s-1]}(X_\ell)$ says exactly that $C_{[s]}(X_\ell) = C_{[s-1]}(X_\ell)$. Since $\alpha_\ell = 0$ pins coordinate s of $\mathbf{w}^*_\ell$ to zero, expanding (4.2) at both depths gives $C^{(s)}_{MM}(X_\ell, \mathbf{w}^*) = C^{(s-1)}_{MM}(X_\ell, \mathbf{w}^*_\ell)$. Through the induction hypothesis and Lemma 4.17,

$$C^{(s)}_{MM}(X, \mathbf{w}^*) \leq \delta \cdot C^{(s-1)}_{MM}(\mathbf{w}^*_\ell) + \beta + \frac{s-1}{k}\epsilon \leq \delta \cdot C^{(s)}_{MM}(\mathbf{w}^*) + \beta + \frac{s}{k}\epsilon,$$

which is the bound we wanted for s.

Case 2: binary search mixture. If the search runs, X is a mixture of X_ℓ and X_r. Lemma 4.18 splits the cost across both endpoints, and Lemma 4.19 bridges each term from

the true vector $\mathbf{w}^*$ back to its respective boundary vector:

$$C_{MM}^{(s)}(X, \mathbf{w}^*) = p \cdot C_{MM}^{(s-1)}(X_\ell, \mathbf{w}_\ell^*) + (1-p) \cdot C_{MM}^{(s-1)}(X_r, \mathbf{w}_r^*) + \underbrace{p(w_s^* - \alpha_\ell) \cdot slack_\ell + (1-p)(\alpha_r - w_s^*) \cdot slack_r}_{E}. \tag{4.3}$$

Via the induction hypothesis and Lemma 4.17, the first two terms are together bounded by $\delta \cdot C_{MM}^{(s)}(\mathbf{w}^*) + \beta + \frac{s-1}{k}\epsilon$. To bound the residual term E, substitute the mixture formula $p = \frac{slack_r}{slack_\ell + slack_r}$; the unknown target weight w_s^* cancels out entirely:

$$\begin{aligned} E &= \frac{slack_\ell \cdot slack_r}{slack_\ell + slack_r}(w_s^* - \alpha_\ell) + \frac{slack_\ell \cdot slack_r}{slack_\ell + slack_r}(\alpha_r - w_s^*) \\ &= \frac{slack_\ell \cdot slack_r}{slack_\ell + slack_r}(\alpha_r - \alpha_\ell) \\ &= error. \end{aligned}$$

By the binary search stopping criterion, $error \leq \epsilon/k$. Adding this budget completes the inductive step. □

Completing the Proof

The inductive bound holds for every level s, including the top level $s = k$ we actually care about.

Proof of Theorem 4.7. At $s = k$, the interpolating function collapses to the objective we started with: $C_{MM}^{(k)}(Y, \mathbf{u}) = C_{[k]}(Y)$, independently of $\mathbf{u}$. Thus,

$$C_{MM}^{(k)}(\mathbf{w}^*) = \inf_{Y \in \mathcal{F}} \max_x C_x(Y) = C_{OPT}.$$

Plugging $s = k$ into Lemma 4.20 gives $C_{[k]}(X) \leq \delta \cdot C_{OPT} + \beta + \epsilon$, which is precisely the $(\delta, \beta + \epsilon)$-approximation the theorem promises.

For the running time, the recursion spans $k - 1$ levels. Each level makes 2 Phase-1 calls

and up to $\lceil \log(kL_c/(2\epsilon)) \rceil$ Phase-2 calls, giving a branching factor of at most $\log(4kL_c/\epsilon)$. A tree of depth $k-1$ with this branching factor has at most $\log^{k-1}(4kL_c/\epsilon)$ leaves, and each leaf costs one call to ALG, giving the total running time $T_{\text{ALG}} \cdot \log^{k-1}(4kL_c/\epsilon)$. □

4.4 The SHERWOOD Algorithm

We now present the second algorithm that realizes Theorem 4.8. As mentioned in Section 4.2, SHERWOOD adapts the classical *Hedge* algorithm [Freund and Schapire, 1999, Arora et al., 2012] (also known as Multiplicative Weights Update, or MWU) to our framework. It applies Hedge to the zero-sum game between WMCM and WACM from Section 4.1, with ALG acting as the best-response oracle.

Hedge is attractive here for its simplicity: the update rule is a one-line exponential reweighting, and the algorithm fits in a few lines of pseudocode. Section 4.4.1 describes the algorithm in detail, and Section 4.4.2 proves Theorem 4.8.

4.4.1 The Algorithm

SHERWOOD runs for $T = \lceil L_c^2 \ln k/(2\epsilon^2) \rceil$ rounds with learning rate $\eta = 2\sqrt{2\ln k/(L_c^2 T)}$, starting from the uniform weight vector $u_x^{(1)} = 1/k$. In each round t, it hands the current weights $\mathbf{u}^{(t)}$ to ALG, which returns a solution $Y^{(t)}$ approximately minimizing the weighted cost $\sum_x u_x^{(t)} C_x(Y)$. It reads off the cost every agent incurs on $Y^{(t)}$ and updates the weights multiplicatively,

$$w_x^{(t+1)} = u_x^{(t)} \cdot \exp\big(\eta \cdot C_x(Y^{(t)})\big), \qquad u_x^{(t+1)} = w_x^{(t+1)} \Big/ \sum_{j=1}^{k} w_j^{(t+1)}.$$

After T rounds, it returns the uniform mixture $\bar{Y}$ over all the solutions queried, $Y^{(1)}, \ldots, Y^{(T)}$. The complete pseudocode is given in Algorithm 3.

Algorithm 3 SHERWOOD(ϵ): Hedge applied to ALG

Require: Precision $\epsilon > 0$, agents $A_1, \dots, A_k$ with cost functions $C_1, \dots, C_k \in [0, L_c]$
Ensure: A distribution $\bar{Y} \in \mathcal{F}$ approximating the optimal WMCM solution

$T \leftarrow \lceil L_c^2 \ln k \,/\, (2\epsilon^2) \rceil$
$\eta \leftarrow 2\sqrt{2 \ln k \,/\, (L_c^2 T)}$
Initialize $u_x^{(1)} \leftarrow 1/k$ for all $x \in \{1, \dots, k\}$
for $t = 1, \dots, T$ **do**
    Run ALG with weights $\mathbf{u}^{(t)}$ to obtain solution $Y^{(t)}$
    Observe costs $C_1(Y^{(t)}), \dots, C_k(Y^{(t)})$
    for $x = 1, \dots, k$ **do** ▷ Multiplicative weight update
        $w_x^{(t+1)} \leftarrow u_x^{(t)} \cdot \exp\big(\eta \cdot C_x(Y^{(t)})\big)$
    end for
    Normalize: $u_x^{(t+1)} \leftarrow w_x^{(t+1)} / \sum_{j=1}^{k} w_j^{(t+1)}$ for all x
end for
return $\bar{Y}$: draw uniformly from $\{Y^{(1)}, \dots, Y^{(T)}\}$

Discussion. T and η are chosen so that Hedge's regret, averaged over the run, falls below ϵ (Section 4.4.2 derives both values). If η is too small the weights barely move, and if it is too large they overshoot. The multiplicative update boosts agents whose cost on $Y^{(t)}$ was high and suppresses agents whose cost was low. Over many rounds this shifts weight toward whichever agents ALG keeps neglecting, so its later responses have to take them more seriously. Each agent's expected cost under $\bar{Y}$ is its average cost over the run, $C_x(\bar{Y}) = \frac{1}{T} \sum_{t=1}^{T} C_x(Y^{(t)})$. Hedge's regret bound limits how far this average can fall behind the best fixed weighting chosen with hindsight, and the proof below turns that bound into a guarantee on the worst agent's cost under $\bar{Y}$.

4.4.2 Analysis

Proof of Theorem 4.8. We prove that SHERWOOD returns a $(\delta, \beta + \epsilon)$-approximation of the WMCM problem. The proof proceeds in three steps: (1) we invoke the standard Hedge regret bound, (2) we relate each round's weighted cost to the optimal WMCM value via ALG's guarantee, and (3) we bound the maximum agent cost of the output distribution and set T to achieve the desired precision.

Hedge regret bound. Run with learning rate $\eta = 2\sqrt{2\ln k/(L_c^2 T)}$ against any sequence of reward vectors $\mathbf{C}^{(t)} = (C_1(Y^{(t)}), \ldots, C_k(Y^{(t)})) \in [0, L_c]^k$, Hedge guarantees that for every fixed comparator distribution $\mathbf{u}^* \in \Delta_k$,

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{u}^{(t)}, \mathbf{C}^{(t)}\rangle \;\geq\; \frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{u}^{*}, \mathbf{C}^{(t)}\rangle \;-\; L_c\sqrt{\frac{\ln k}{2T}}, \tag{4.4}$$

where $\langle \mathbf{u}, \mathbf{C}\rangle = \sum_x u_x C_x$. This is the classical Hedge (multiplicative weights) regret bound with the optimally tuned learning rate, proved via a potential-function argument together with Hoeffding's lemma; see, e.g., Cesa-Bianchi and Lugosi [2006, Theorem 2.2], or Freund and Schapire [1999], Arora et al. [2012] for the original multiplicative-weights derivation. We do not reprove it here.

Bounding each round's weighted cost via ALG. At each round t, the algorithm calls ALG with the weight vector $\mathbf{u}^{(t)} \in \Delta_k$. Since ALG is a (δ, β)-approximation of the WACM problem, the cost of the returned solution $Y^{(t)}$ satisfies:

$$\sum_{x=1}^{k} u_x^{(t)} \cdot C_x(Y^{(t)}) \leq \delta \cdot \min_{Y\in\mathcal{F}} \sum_{x=1}^{k} u_x^{(t)} \cdot C_x(Y) + \beta.$$

We now relate the right-hand side to C_{OPT}. Let S^* denote the optimal WMCM distribution, so $\max_x C_x(S^*) = C_{OPT}$. Then for any $\mathbf{u}^{(t)} \in \Delta_k$:

$$\min_{Y\in\mathcal{F}} \sum_{x=1}^{k} u_x^{(t)} \cdot C_x(Y) \leq \sum_{x=1}^{k} u_x^{(t)} \cdot C_x(S^*) \leq \sum_{x=1}^{k} u_x^{(t)} \cdot C_{OPT} = C_{OPT},$$

where the last inequality uses $C_x(S^*) \leq \max_j C_j(S^*) = C_{OPT}$ for all x, and the last equality uses $\sum_x u_x^{(t)} = 1$. Therefore:

$$\langle \mathbf{u}^{(t)}, \mathbf{C}^{(t)}\rangle = \sum_{x=1}^{k} u_x^{(t)} \cdot C_x(Y^{(t)}) \leq \delta \cdot C_{OPT} + \beta. \tag{4.5}$$

Bounding the maximum agent cost of $\bar{Y}$, and setting T. The output distribution $\bar{Y}$ draws uniformly from $\{Y^{(1)}, \ldots, Y^{(T)}\}$, so the expected cost of agent x is:

$$C_x(\bar{Y}) = \frac{1}{T}\sum_{t=1}^{T} C_x(Y^{(t)}).$$

The maximum agent cost can be written as a maximization over Δ_k:

$$\max_{1\le x\le k} C_x(\bar{Y}) = \max_{\mathbf{u}^*\in\Delta_k} \sum_{x=1}^{k} u_x^* \cdot C_x(\bar{Y}) = \max_{\mathbf{u}^*\in\Delta_k} \frac{1}{T}\sum_{t=1}^{T} \langle \mathbf{u}^*, \mathbf{C}^{(t)}\rangle, \tag{4.6}$$

where the first equality holds because the maximum of a linear function over the simplex is attained at a vertex $\mathbf{e}_x$ (the standard basis vector corresponding to the most expensive agent).

By the regret bound (4.4) and the per-round bound (4.5), for any $\mathbf{u}^* \in \Delta_k$:

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{u}^*, \mathbf{C}^{(t)}\rangle \le \frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{u}^{(t)}, \mathbf{C}^{(t)}\rangle + L_c\sqrt{\frac{\ln k}{2T}} \le \delta \cdot C_{OPT} + \beta + L_c\sqrt{\frac{\ln k}{2T}}.$$

Since this holds for all $\mathbf{u}^* \in \Delta_k$, taking the maximum over $\mathbf{u}^*$ and applying (4.6):

$$\max_{1\le x\le k} C_x(\bar{Y}) \le \delta \cdot C_{OPT} + \beta + L_c\sqrt{\frac{\ln k}{2T}}. \tag{4.7}$$

It remains to choose T so that the additive error term $L_c\sqrt{\ln k/(2T)}$ is at most ϵ:

$$L_c\sqrt{\frac{\ln k}{2T}} \le \epsilon \quad\Longleftrightarrow\quad T \ge \frac{L_c^2 \ln k}{2\epsilon^2}.$$

Setting $T = \lceil L_c^2 \ln k/(2\epsilon^2)\rceil$ satisfies this condition. Substituting into (4.7):

$$\max_{1\le x\le k} C_x(\bar{Y}) \le \delta \cdot C_{OPT} + \beta + \epsilon.$$

This shows that $\bar{Y}$ is a $(\delta, \beta+\epsilon)$-approximation of the WMCM problem.

Running time. Each of the T rounds requires one call to ALG (taking time T_{ALG}) plus $O(k)$ time for the weight update and normalization. Thus the total running time is $T \cdot (T_{\text{ALG}} + O(k)) = T_{\text{ALG}} \cdot \lceil L_c^2 \ln k/(2\epsilon^2) \rceil + O(k \cdot L_c^2 \ln k/\epsilon^2)$. Since T_{ALG} typically dominates the $O(k)$ update cost, the overall complexity is $T_{\text{ALG}} \cdot \lceil L_c^2 \ln k/(2\epsilon^2) \rceil$ as stated in Theorem 4.8. □

4.5 The MARIAN Algorithm

The third algorithm for the WMCM problem is named MARIAN, after Maid Marian, Robin Hood's partner in strategy. While SHERWOOD approaches the problem through online learning, MARIAN takes a different route through convex geometry: it solves the dual optimization problem using the ellipsoid method [Khachiyan, 1979, Grötschel et al., 1988].

MARIAN closes the gap the other two leave open: ROBIN's search is exponential in k, and SHERWOOD's is polynomial in $1/\epsilon$, but MARIAN achieves a running time that is polynomial in k *and* logarithmic in $1/\epsilon$. The key is to use more of what ALG already computes: at any weights $\mathbf{u}$, the solution ALG returns certifies a global majorant on the min-max objective (Section 4.5.1), which is used to reduce the search space. The same construction underlies García-Soriano and Bonchi's maxmin-fair combinatorial search [García-Soriano and Bonchi, 2021] (Section 2.5.2); here it slots into our general (δ, β)-approximate-oracle framework, so any problem that reduces to WACM inherits the ellipsoid method's precision for free.

4.5.1 The Algorithm

MARIAN maintains an ellipsoid $E^{(t)}$, initialized as the minimum-volume ellipsoid containing Δ_k. At each round t, it takes the center $\mathbf{u}^{(t)}$ of $E^{(t)}$. If $\mathbf{u}^{(t)} \notin \Delta_k$, it cuts $E^{(t)}$ with a separating hyperplane between $\mathbf{u}^{(t)}$ and Δ_k. Otherwise, it calls ALG at weights $\mathbf{u}^{(t)}$ to obtain a solution $Y^{(t)}$, observes the resulting cost vector $\mathbf{c}^{(t)} = (C_1(Y^{(t)}), \ldots, C_k(Y^{(t)}))$, i.e., $Y^{(t)}$'s cost to each agent, updates the best value $\langle \mathbf{u}^{(t)}, \mathbf{c}^{(t)} \rangle$ seen so far if it improves on it, and cuts $E^{(t)}$ with the half-space $\{\mathbf{u} : \langle \mathbf{c}^{(t)}, \mathbf{u} - \mathbf{u}^{(t)} \rangle \geq 0\}$. Either way, it re-encloses what remains in a new

minimum-volume ellipsoid $E^{(t+1)}$. After $T = \lceil 2k(k+1)\ln(2kL_c/\epsilon)\rceil$ rounds, it solves an auxiliary linear program over the T queried solutions and returns the resulting distribution $\bar{Y}$.

The complete pseudocode is given in Algorithm 4.

Algorithm 4 MARIAN(ϵ)

Require: Precision $\epsilon > 0$, agents $A_1, \ldots, A_k$ with cost functions $C_1, \ldots, C_k$
Ensure: A distribution $\bar{Y} \in \mathcal{F}$ approximating the optimal WMCM solution

$T \leftarrow \lceil 2k(k+1)\ln(2kL_c/\epsilon)\rceil$
Initialize ellipsoid $E^{(0)}$ as the minimum-volume ellipsoid containing Δ_k
$f_{\text{best}} \leftarrow -\infty$
for $t = 0, 1, \ldots, T-1$ **do**
    Let $\mathbf{u}^{(t)}$ be the center of $E^{(t)}$
    if $\mathbf{u}^{(t)} \notin \Delta_k$ **then** ▷ Project back to simplex
        Use a separating hyperplane between $\mathbf{u}^{(t)}$ and Δ_k to cut $E^{(t)}$
    else
        Run ALG with weights $\mathbf{u}^{(t)}$ to obtain solution $Y^{(t)}$
        Observe cost vector $\mathbf{c}^{(t)} = (C_1(Y^{(t)}), \ldots, C_k(Y^{(t)}))$
        $f^{(t)} \leftarrow \langle \mathbf{u}^{(t)}, \mathbf{c}^{(t)} \rangle$
        if $f^{(t)} > f_{\text{best}}$ **then**
            $f_{\text{best}} \leftarrow f^{(t)}$
        end if
        Cut with the majorant: keep $\{\mathbf{u} : \langle \mathbf{c}^{(t)}, \mathbf{u} - \mathbf{u}^{(t)} \rangle \geq 0\}$
    end if
    $E^{(t+1)} \leftarrow$ minimum-volume ellipsoid containing $E^{(t)} \cap$ half-space
end for
Solve the auxiliary LP: $\min v$ s.t. $\sum_t p_t\, C_x(Y^{(t)}) \leq v$ for all x, $\sum_t p_t = 1$, $p_t \geq 0$
return $\bar{Y}$: draw $Y^{(t)}$ with probability p_t^*

Discussion. Recall from Corollary 4.4 that the WMCM problem is equivalent to the saddle-point problem $C_{OPT} = \max_{\mathbf{u}\in\Delta_k} f(\mathbf{u})$, where $f(\mathbf{u}) = \min_{Y\in\mathcal{F}} \sum_x u_x C_x(Y)$. The function f is concave (as a pointwise minimum of linear functions) and piecewise linear. Maximizing a concave function over the simplex Δ_k (a convex body in $\mathbb{R}^{k-1}$) is a problem the ellipsoid method is well suited for, converging in $O(k^2 \log(kL_c/\epsilon))$ iterations (proved in Section 4.5.2 below) even though f is not smooth.

A plain call to ALG is enough to drive this search, because its output carries more than a single number. At weights $\mathbf{u}$, ALG returns a cost vector $\mathbf{c} = (C_1(Y^*), \ldots, C_k(Y^*))$, for

$Y^* = \textsc{Alg}(\mathbf{u})$, and this cost vector upper-bounds f everywhere: $f(\mathbf{u}') \le \langle \mathbf{c}, \mathbf{u}' \rangle$ for all $\mathbf{u}'$, simply because Y^* is one particular feasible solution. If Alg solved Wacm exactly, this hyperplane would be tangent to f at $\mathbf{u}$. Since Alg is in general only a (δ, β)-approximation, we only get this weaker, but still always true, global majorant, and that is what is used to cut down the search space at every round. Section 4.5.2 shows this majorant property alone is enough to drive the ellipsoid method's cutting planes.

The separating hyperplane used when $\mathbf{u}^{(t)} \notin \Delta_k$ is guaranteed to exist by the hyperplane separation theorem (a point outside a closed convex set can always be separated from it), and is easy to construct here. Since Δ_k is cut out of the affine hyperplane $\sum_x u_x = 1$ purely by the nonnegativity constraints $u_x \ge 0$, a center violating Δ_k must have some coordinate $u_j^{(t)} < 0$, and the hyperplane $u_j = 0$ already separates it from all of Δ_k. This coordinate can be found in $O(k)$ time by scanning $\mathbf{u}^{(t)}$.

4.5.2 Analysis

Throughout, write $d = k - 1$ for the dimension of Δ_k as an affine subset of $\mathbb{R}^k$, and recall that L_c upper-bounds every agent's cost, so $0 \le C_x(Y) \le L_c$ for all x and all $Y \in \mathcal{F}$.

Two classical ellipsoid facts are needed first. The first bounds the volume of the initial ellipsoid.

Lemma 4.21 (John Ellipsoid Bound [Grötschel et al., 1988])**.** *The minimum-volume ellipsoid $E^{(0)}$ containing Δ_k (Algorithm 4's initial ellipsoid) satisfies* $\mathrm{vol}(E^{(0)}) \le d^d \,\mathrm{vol}(\Delta_k)$*, where both volumes are d-dimensional, taken within the affine hull of Δ_k.*

The second is the source of the ellipsoid method's power: with each divide-and-cover step, the volume of the ellipsoid shrinks by a constant factor.

Lemma 4.22 (Ellipsoid Volume Reduction [Grötschel et al., 1988])**.** *Let $E \subset \mathbb{R}^d$ be an ellipsoid with center $\mathbf{c}$. For any half-space $H = \{\mathbf{x} : \langle \mathbf{a}, \mathbf{x} \rangle \ge \langle \mathbf{a}, \mathbf{c} \rangle\}$, the minimum-volume*

ellipsoid E' containing $E \cap H$ satisfies:

$$\frac{\mathrm{vol}(E')}{\mathrm{vol}(E)} \leq e^{-1/(2(d+1))}.$$

The next lemma justifies MARIAN's choice of cutting plane.

Lemma 4.23 (Global Majorant)**.** *Let $\mathbf{u} \in \Delta_k$, let $Y^* = \textsc{Alg}(\mathbf{u})$, and let*

$$\mathbf{c} = (C_1(Y^*), \ldots, C_k(Y^*)).$$

Then for every $\mathbf{u}' \in \Delta_k$,

$$f(\mathbf{u}') \leq \langle \mathbf{c}, \mathbf{u}' \rangle.$$

Proof. Since $Y^* \in \mathcal{F}$ is a feasible solution, $f(\mathbf{u}') = \min_{Y \in \mathcal{F}} \sum_x u'_x C_x(Y) \leq \sum_x u'_x C_x(Y^*) = \langle \mathbf{c}, \mathbf{u}' \rangle$. □

Unlike in the idealized case where ALG solves WACM exactly, Lemma 4.23 does not claim that $\mathbf{c}$ is tangent to f at $\mathbf{u}$. It only claims that the affine function $\mathbf{u}' \mapsto \langle \mathbf{c}, \mathbf{u}' \rangle$ lies above f everywhere on Δ_k. This weaker, but always true, property is all the proof below uses; it makes no assumption on ALG's approximation quality beyond Definition 4.5.

The following lemma shows that no matter which weight assignment $\mathbf{u} \in \Delta_k$ we choose, at least one of the queried solutions $Y^{(1)}, \ldots, Y^{(T)}$ already achieves the $(\delta, \beta + \epsilon)$-approximation bound we are after.

Lemma 4.24. *For every $\mathbf{u} \in \Delta_k$,*

$$\min_{t<T} \langle \mathbf{u}, \mathbf{c}^{(t)} \rangle \leq \delta \cdot C_{OPT} + \beta + \epsilon.$$

Proof. Write $\tau := \delta \cdot C_{OPT} + \beta + \epsilon$ and, for each t, $\mathbf{c}^{(t)} = C(Y^{(t)})$ for the cost vector observed at round t. Since ALG is a (δ, β)-approximation of WACM and $f(\mathbf{u}^{(t)}) \leq \max_{\mathbf{u} \in \Delta_k} f(\mathbf{u}) = C_{OPT}$ (Section 4.5.1), every query satisfies $\langle \mathbf{u}^{(t)}, \mathbf{c}^{(t)} \rangle \leq \delta \cdot f(\mathbf{u}^{(t)}) + \beta \leq \delta \cdot C_{OPT} + \beta = \tau - \epsilon$.

Suppose, toward a contradiction, that some $\mathbf{u}' \in \Delta_k$ satisfies $\langle \mathbf{u}', \mathbf{c}^{(t)} \rangle > \tau$ for every $t < T$. For $r = \epsilon/\tau \in (0, 1]$, consider the shrunk simplex $S_r := (1-r)\mathbf{u}' + r\Delta_k \subseteq \Delta_k$, which has volume $\mathrm{vol}(S_r) = r^d \,\mathrm{vol}(\Delta_k)$. For any $\mathbf{u} = (1-r)\mathbf{u}' + r\mathbf{v} \in S_r$ with $\mathbf{v} \in \Delta_k$, nonnegativity of costs gives $\langle \mathbf{u}, \mathbf{c}^{(t)} \rangle \geq (1-r)\langle \mathbf{u}', \mathbf{c}^{(t)} \rangle > (1-r)\tau = \tau - \epsilon \geq \langle \mathbf{u}^{(t)}, \mathbf{c}^{(t)} \rangle$ for every $t < T$. So every point of S_r satisfies the cutting condition $\langle \mathbf{c}^{(t)}, \mathbf{u} - \mathbf{u}^{(t)} \rangle \geq 0$ at each round where $\mathbf{u}^{(t)} \in \Delta_k$, and when $\mathbf{u}^{(t)} \notin \Delta_k$, the separating-hyperplane cut retains all of $\Delta_k \supseteq S_r$ by construction. By induction on Lemma 4.22's update rule, $S_r \subseteq E^{(t)}$ for every $t \leq T$, so $\mathrm{vol}(E^{(T)}) \geq \mathrm{vol}(S_r) = (\epsilon/\tau)^d \,\mathrm{vol}(\Delta_k)$.

But by Lemma 4.22 applied T times and Lemma 4.21, $\mathrm{vol}(E^{(T)}) \leq \mathrm{vol}(E^{(0)}) \cdot e^{-T/(2(d+1))} \leq d^d \,\mathrm{vol}(\Delta_k) \cdot e^{-T/(2k)}$. Combining the two bounds and cancelling $\mathrm{vol}(\Delta_k)$ gives $(\epsilon/\tau)^d \leq d^d e^{-T/(2k)}$, i.e. $T \leq 2kd \ln(d\tau/\epsilon)$. Since $d = k - 1 < k + 1$ and $\tau = \delta \cdot C_{OPT} + \beta + \epsilon \leq \delta L_c + \beta + \epsilon = O(L_c)$ (as $C_{OPT} \leq L_c$ and δ, β are constants), this bound is dominated by $2k(k+1)\ln(2kL_c/\epsilon)$ (the value of T used in Algorithm 4) for a suitable choice of the constant hidden in $O(L_c)$. This contradicts T's actual value, so no such $\mathbf{u}'$ exists. □

Proof of Theorem 4.9. Let $\tau := \delta \cdot C_{OPT} + \beta + \epsilon$; we show Algorithm 4 returns a distribution $\bar{Y}$ with $\max_x C_x(\bar{Y}) \leq \tau$.

From dual to primal. The auxiliary LP of Algorithm 4 computes

$$v^* = \min_{\mathbf{p} \in \Delta_T} \max_x \sum_t p_t C_x(Y^{(t)});$$

it has T variables and $k+1$ constraints, solvable in $O(k^2 T)$ time. Writing $\max_x(\cdot) = \max_{\mathbf{u} \in \Delta_k} \sum_x u_x(\cdot)$ (the maximum of a linear function over Δ_k is attained at a vertex) turns this into a bilinear objective over the two simplices Δ_T and Δ_k:

$$v^* = \min_{\mathbf{p} \in \Delta_T} \max_{\mathbf{u} \in \Delta_k} \sum_t p_t \langle \mathbf{u}, \mathbf{c}^{(t)} \rangle.$$

By Theorem 4.3, the two optimizations can be exchanged:

$$v^* = \max_{\mathbf{u}\in\Delta_k} \min_{\mathbf{p}\in\Delta_T} \sum_t p_t \langle \mathbf{u}, \mathbf{c}^{(t)} \rangle = \max_{\mathbf{u}\in\Delta_k} \min_{t<T} \langle \mathbf{u}, \mathbf{c}^{(t)} \rangle \le \tau,$$

where the middle equality holds because $\min_{\mathbf{p}\in\Delta_T} \sum_t p_t a_t = \min_t a_t$ for any reals a_t, and the inequality is Lemma 4.24.

Drawing $Y^{(t)}$ with probability p_t^* for an LP-optimal $\mathbf{p}^*$ therefore produces $\bar{Y} \in \mathcal{F}$ with $\max_x C_x(\bar{Y}) = v^* \le \tau$, as required.

Time complexity. As established in the proof of Lemma 4.24, $T = O(k^2 \log(kL_c/\epsilon))$ iterations suffice, each costing one call to ALG plus $O(k^2)$ arithmetic for the ellipsoid update or separating-hyperplane test. The final LP takes $O(k^2 T)$ time. The total running time is $T_{\text{ALG}} \cdot O(k^2 \log(kL_c/\epsilon))$. □

Remark 4.25. MARIAN trades SHERWOOD's logarithmic dependence on k (i.e., $\ln k$) for a *polynomial* dependence (k^2), but gains an exponential improvement in precision dependence: $\log(1/\epsilon)$ versus $1/\epsilon^2$. For any fixed number of agents, MARIAN dominates SHERWOOD for sufficiently small ϵ.

4.6 Practical Considerations

The reason MARIAN scales better than SHERWOOD is not that it needs anything extra from ALG: like the other two algorithms, it only ever calls ALG and reads off the cost vector it returns. The difference is how much of that vector it uses. SHERWOOD treats each cost vector as a plain reward signal for its multiplicative-weight update, while MARIAN recognizes that the same vector is also a global majorant on the min-max objective (Section 4.5.1), a whole supporting hyperplane rather than a single number, and feeds that directly into the ellipsoid method to cut the search space much faster.

These asymptotic bounds do not settle which algorithm is fastest in practice, however. The ellipsoid method is numerically delicate: its ellipsoids can become badly conditioned over many iterations, and it tends to run close to its worst-case bound rather than beating it, unlike many algorithms whose typical-case behavior is much better than their worst case. SHERWOOD's multiplicative update, by contrast, is a handful of floating-point operations per round with no matrices to maintain, so its constant factors are small and its numerical behavior is stable. Even though MARIAN wins asymptotically, a real implementation with many agents and only moderate precision requirements may still be better served by SHERWOOD, simply because it tends to be faster and more predictable in wall-clock time.

Chapter 5

Conclusion

In this thesis, we studied the problem of designing fair algorithms for multi-agent environments, formalized as *Weighted Max Cost Minimization* (WMCM): minimizing the maximum weighted cost across all agents sharing a common solution, so that no single agent bears a disproportionate share of the cost.

Chapter 3 was devoted to fairness for shared binary search trees, where k agents with different access distributions must agree on a single tree. We measured the cost of sharing in two ways: additively, as extra comparisons paid per access, and multiplicatively, as a factor over each agent's own optimal cost. For additive optimality, we gave a simple algorithm, FUSE, with an almost matching lower bound. For multiplicative optimality we gave three algorithms of increasing strength: COMB, which is worst-case optimal; BLEND, which goes beyond the worst case by adapting to each instance's cost scales; and TUNE, which is asymptotically optimal for arbitrary access distributions. We also showed how to make every construction robust to inaccurate predictions of the agents' access patterns, at a reasonable cost.

We then generalized beyond BSTs (Chapter 4) to arbitrary optimization problems shared by k agents. Since any deterministic algorithm that approximates WACM well can only guarantee a k times worse approximation for WMCM, we gave a framework that converts any (δ, β)-approximation algorithm for the corresponding weighted-sum problem, *Weighted*

Average Cost Minimization (WACM), into a randomized $(\delta, \beta + \epsilon)$-approximation algorithm for WMCM, closing this gap almost entirely. We gave three such conversions with different trade-offs: ROBIN, a recursive binary search over agent weights; SHERWOOD, based on the Hedge algorithm from online learning; and MARIAN, based on the ellipsoid method, which improves on the other two by running in time polynomial in k and only logarithmic in $1/\epsilon$.

5.1 Future Directions

We showed the price of fairness to be $\log k$ for shared BSTs. Whether the same factor recurs for the other problems our framework applies to, such as shared minimum spanning trees, matching, and clustering, or is specific to the structure of BSTs, is an open question. Each of these problems likely also admits sharper, problem-specific bounds and WACM solvers beyond what the generic framework guarantees. This includes combinatorial problems we have not considered, such as facility location or network design.

More speculatively, the framework as presented assumes all agents and their cost functions are fixed and known in advance; extending it to online or dynamic settings, where agents arrive over time or their costs change, and incorporating learned predictions about agent behavior in the spirit of Chapter 2, are both natural directions we leave unexplored.

Another open question worth digging into is algorithmic: MARIAN already breaks SHERWOOD's $1/\epsilon^2$ precision barrier by exploiting the supergradient carried by ALG's output, and its LP-dual refinement keeps the resulting output distribution compact, but whether its k^2 dependence on the number of agents can be reduced while keeping $\log(1/\epsilon)$ precision remains open.

Bibliography

Alessandro Agnetis, Pitu B Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Fair correlation clustering. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108, pages 4195–4205, 2020.

Elad Aigner-Horev and Erel Segal-Halevi. Envy-free matchings in bipartite graphs and their applications to fair division. *Information Sciences*, 587:164–187, 2022.

Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

Makis Arsenis and Robert Kleinberg. Individual fairness in prophet inequalities. In *Proceedings of the 23rd ACM Conference on Economics and Computation (EC)*, 2022.

Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger. Multi-tenant databases for software as a service: Schema-mapping techniques. In Jason Tsong-Li Wang, editor, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1195–1206. ACM, 2008. doi: 10.1145/1376616.1376736.

Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. Curran Associates, Inc., 2020.

Siddhartha Banerjee, Vasilis Gkatzelis, Safwan Hossain, Billy Jin, Evi Micha, and Nisarg Shah. Proportionally fair online allocation of public goods with predictions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 20–28, 2023.

Nikhil Bansal and Maxim Sviridenko. The Santa Claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2006.

Paul J Bayer. Improved bounds on the costs of optimal and balanced binary search trees. Technical Report MAC TM-69, Massachusetts Institute of Technology, Project MAC, 1975.

Ziyad Benomar and Christian Coester. Learning-augmented priority queues. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4955–4966, 2019.

Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 18:1–18:22, 2019.

Dimitris Bertsimas, Vivek F Farias, and Nikolaos Trichakis. The price of fairness. *Operations Research*, 59(1):17–31, 2011.

Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, and Maria Kyropoulou. The efficiency of fair division. *Theory of Computing Systems*, 50:589–610, 2012.

Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Jingbang Chen, Xinyuan Cao, Alicia Stepin, and Li Chen. On the power of learning-augmented search trees. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 7739–7762. PMLR, 2025.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5029–5037, 2017.

José Correa, Andrés Cristi, Laurent Feuilloley, Tim Oosterwijk, and Alexandros Tsigonias-Dimitriadis. The secretary problem with independent sampling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2047–2058, 2021.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

Mohammadreza Daneshvaramoli, Helia Karisani, Mohammad Hajiesmaili, Shahin Kamali, and Cameron Musco. Fairness in the k-server problem. *arXiv preprint arXiv:2512.20960*, 2025.

Lee D. Davisson. Universal noiseless coding. *IEEE Transactions on Information Theory*, 19 (6):783–795, 1973.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, pages 205–220. ACM, 2007.

Ilias Diakonikolas and Mihalis Yannakakis. Small approximate Pareto sets for biobjective shortest paths and other problems. *SIAM Journal on Computing*, 39(4):1340–1371, 2009.

Emily Diana, Wesley Gill, Michael Kearns, Krishnaram Kenthapadi, and Aaron Roth. Minimax group fairness: Algorithms and experiments. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, pages 66–76, 2021.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, pages 214–226, 2012.

Matthias Ehrgott. *Multicriteria optimization*. Springer, 2005.

Seyed A. Esmaeili, Brian Brubach, Aravind Srinivasan, and John Dickerson. Fair clustering under a bounded cost. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 14345–14357, 2021.

Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 3123–3132. PMLR, 2020.

Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

Robert G. Gallager. Source coding with side information and universal coding. Technical Report LIDS-P-937, Laboratory for Information and Decision Systems, MIT, 1976.

David García-Soriano and Francesco Bonchi. Fair-by-design matching. *Data Mining and Knowledge Discovery*, 34:1291–1335, 2020.

David García-Soriano and Francesco Bonchi. Maxmin-fair ranking: Individual fairness under group-fairness constraints. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 436–446, 2021.

Mehrdad Ghadiri, Samira Samadi, and Santosh Vempala. Socially fair k-means clustering. In *ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, pages 438–448, 2021.

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In David G. Andersen and Sylvia Ratnasamy, editors, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2011.

Christian Glaßer, Christian Reitwißner, Heinz Schmitz, and Maximilian Witek. Hardness and approximability in multi-objective optimization. *Electronic Colloquium on Computational Complexity*, 17(31), 2010.

Ashish Goel and Adam Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Algorithmica*, 44:301–323, 2006.

Dishant Goyal and Ragesh Jaiswal. Tight FPT approximation for socially fair clustering. *Information Processing Letters*, 182:106383, 2023.

Ronald L Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

Swati Gupta, Jai Moondra, and Mohit Singh. Which l_p norm is the fairest? Approximations for fair facility location across all "p". In *Proceedings of the 24th ACM Conference on Economics and Computation (EC)*, 2023.

Horst W Hamacher and Gerhard Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3323–3331, 2016.

Lisa Hellerstein, Thomas Lidbetter, and Daniel Pirutinsky. Solving zero-sum games using best response oracles with applications to search games. *arXiv preprint arXiv:1704.02657*, 2017.

Hadi Hosseini, Zhiyi Huang, Ayumi Igarashi, and Nisarg Shah. Class fairness in online matching. In *AAAI Conference on Artificial Intelligence*, pages 5673–5680, 2023.

Sungjin Im and Benjamin Moseley. Fair scheduling via iterative quasi-uniform sampling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2601–2615, 2017.

James E. Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998.

Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979.

Matthias Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning (ICML)*, pages 3448–3457, 2019a.

Matthias Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. Guarantees for spectral clustering with fairness constraints. In *International Conference on Machine Learning (ICML)*, pages 3458–3467, 2019b.

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

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 489–504. ACM, 2018. doi: 10.1145/3183713.3196909.

Mayuresh Kunjir, Brandon Fain, Kamesh Munagala, and Shivnath Babu. ROBUS: Fair cache allocation for data-parallel workloads. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, pages 219–234. ACM, 2017. doi: 10.1145/3035918.3064018.

Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

Adam Lechowicz, Rik Sengupta, Bo Sun, Shahin Kamali, and Mohammad Hajiesmaili. Time fairness in online knapsack problems. In *International Conference on Learning Representations (ICLR)*, 2024.

Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1-3):259–271, 1990.

Honghao Lin, Tian Luo, and David Woodruff. Learning augmented binary search trees. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan

Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 13431–13440. PMLR, 2022.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3296–3305. PMLR, 2018.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):1–25, 2021.

Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In *Conference on Learning Theory (COLT)*, pages 3246–3264, 2021.

Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

Natalia Martinez, Martin Bertran, and Guillermo Sapiro. Minimax Pareto fairness: A multi objective perspective. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 6755–6764, 2020.

Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. Online list labeling with predictions. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 60278–60290. Curran Associates, Inc., 2023.

Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975.

Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984.

Neri Merhav and Meir Feder. A strong version of the redundancy–capacity theorem of universal coding. *IEEE Transactions on Information Theory*, 41(3):714–722, 1995.

Neri Merhav and Meir Feder. Universal prediction. *IEEE Transactions on Information Theory*, 44(6):2124–2147, 1998.

Evi Micha and Nisarg Shah. Proportionally fair clustering revisited. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 85:1–85:16, 2020.

Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

John F Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

Minh Hieu Nguyen, Mourad Baiou, Viet Hung Nguyen, and Thi Quynh Trang Vo. Proportional fairness for combinatorial optimization. In *Latin American Symposium on Theoretical Informatics (LATIN)*, volume 14579 of *Lecture Notes in Computer Science*, pages 304–319, 2024.

Christos H Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 86–92. IEEE, 2000.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9684–9693, 2018.

Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.

John Rawls. *A Theory of Justice*. Harvard University Press, 1971.

Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 96:1–96:14, 2018.

Salvatore Sanfilippo. Redis. `https://redis.io`, 2009.

Govind S Sankar, Anand Louis, Meghana Nasre, and Prajakta Nimbhorkar. Matchings with group fairness constraints: Online and offline algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 377–383, 2021.

Daman Deep Singh, Amit Kumar, and Abhijnan Chakraborty. Towards fairness in online service with k servers and its application on fair food delivery. In *AAAI Conference on Artificial Intelligence*, pages 19929–19936, 2024.

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

Vasilis Syrgkanis, Alekh Agarwal, Haipeng Luo, and Robert E. Schapire. Fast convergence of regularized learning in games. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2989–2997, 2015.

Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73(3):291–341, 1996.

Sergei Vassilvitskii and Mihalis Yannakakis. Efficiently computing succinct trade-off curves. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1201–1213. Springer, 2004.

John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

Abraham Wald. Generalization of a theorem by v. Neumann concerning zero sum two person games. *Annals of Mathematics*, 46(2):281–286, 1945.

Arthur Warburton. Approximation of Pareto optima in multiple-objective shortest-path problems. *Operations Research*, 35(1):70–79, 1987.

Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. Curran Associates, Inc., 2020.

Qingyun Wu and Alvin E. Roth. The lattice of envy-free matchings. *Games and Economic Behavior*, 109:201–211, 2018.

Ali Zeynali, Shahin Kamali, and Mohammad Hajiesmaili. Robust learning-augmented dictionaries. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

Statement on the Use of Generative AI

Generative AI tools were used during the preparation of this thesis solely for language editing and polishing of already-drafted text (grammar, clarity, and phrasing). All technical content, ideas, proofs, and analysis are the author's own.