Machine learning in metrical task systems and other on-line problems

Carl Burch

May 2000

CMU-CS-00-135

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Thesis Committee
Avrim Blum, chair
Allan Borodin
Bruce Maggs
Daniel Sleator

This research was sponsored by the National Science Foundation (NSF) under various grants and fellowship awards. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the NSF or the U.S. government.

Keywords: Competitive ratio, metrical task systems, machine learning theory, on-line algorithms

Abstract

We establish and explore a new connection between two general on-line scenarios deriving from two historically disjoint communities. Though the problems are inherently similar, the techniques and questions developed for these two scenarios are very different. From competitive analysis comes the problem of *metrical task systems*, where the algorithm is to decide in which state to process each of several sequential tasks, where each task specifies the processing cost in each state, and the algorithm must pay according to a metric to move between states. And from machine learning comes the problem of *predicting from expert advice* — that is, of choosing one of several experts for each query in a sequence without doing much worse than the best expert overall.

The dissertation includes four results touching on this connection. We begin with the first metrical task system algorithm that can guarantee for every task sequence that the ratio of its expected cost to the cheapest way to process the sequence is only polylogarithmic in the number of states. Then we see how we can use expert-advice results to combine on-line algorithms on-line if there is a fixed cost for changing between the on-line algorithms. The third result establishes new expert-advice algorithms deriving from metrical task system research; in addition to establishing theoretical bounds, we compare the algorithms empirically on a process migration scenario. Finally, we investigate a modified version of paging, where we want to do well against an adversary who is allowed to ignore a paging request cheaply.

Contents

Acknowledgments

Of course there are many people whom I would like to acknowledge for their help in the writing of this thesis — friends and family, students and teachers, mentors and colleagues. I restrict myself to mentioning three, for fear of leaving out people were I to mention more.

The first two are my parents, Charles and Cheri Burch, who taught me much of the background I learned before graduate school, and who persistently motivated me to get back to writing the words (and, more often, the formulas) that appear on these pages.

The third is my advisor Avrim Blum, who has been a model advisor, teaching me most of what I learned as a graduate student and working with me to accomplish that which appears in this thesis. His advice, never peppered with self-interest, has proven very valuable.

Chapter 1

Introduction

1.1 Summary

Beginning in the mid-1980s, researchers of theoretical computer science began investigating the analysis of **on-line algorithms** — that is, algorithms that commit to actions as they receive events. An **on-line problem** defines the types of events and actions that the on-line algorithm can use. Computer technology inspires a wide variety of problems that fall into this framework, including caching, dynamic lists, real-time compression, and call routing.

Researchers soon became interested in abstractions to encompass a variety of on-line problems. Among these were two very prominent problems: the *metrical task system* (MTS) problem [BLS92] and the k-*server problem* [MMS90]. This thesis begins with the metrical task system problem — the simpler of the two — where the algorithm is to decide in which state to process each of several sequential tasks, where each task specifies the processing cost in each state, but changing states also has a cost according to a metric. In particular, we are concerned with how we can use randomization so that regardless of the event sequence, our on-line algorithm's expected cost is not too many times the optimal cost for servicing the sequence.

Independently, in the mid-1990s, researchers interested in machine learning became interested in the following scenario: The on-line learner sees a sequence of examples and wants to predict each example's label before seeing the true label. The hope is that the learner will make few mistakes as it sees more and more examples with their corresponding labels. This is termed the Experts problem.

This thesis demonstrates how one particular problem arising from metrical task systems is intertwined with another particular problem arising from the Experts problem. This connection forms the foundation of this dissertation, on which we build four primary results.*

*Most of the work appearing in this thesis originally appeared in papers by Blum and Burch [BB97]; Bartal, Blum, Burch, and Tomkins [BBBT97]; and Blum, Burch, and Kalai [BBK99]. The author would like to recognize his coauthors, Avrim Blum, Yair Bartal, Andrew Tomkins, and Adam Kalai, who share equally in the development of these concepts. Besides this chapter, Sections 2.1, 2.2, 3.1, and 3.2 describe background material to put this work in context.

A polylogarithmic MTS algorithm

Using on-line learning algorithms, we construct an algorithm guaranteeing that the ratio of its expected cost to the optimal cost (were we to know the entire sequence in advance) is only polylogarithmic in the number of states. In particular, our algorithm guarantees that the on-line algorithm's expected cost is no more than $O(\log^7 n \log\log n)$ times the optimal cost knowing the sequence in advance. Using a much less intuitive technique originating from more traditional on-line algorithms research, we can guarantee an $O(\log^5 n \log\log n)$ bound. These represent the historically first polylogarithmic guarantees for the metrical task system problem. (By refining these techniques further, Fiat and Mendel describe an algorithm guaranteeing an expected cost of at most $O(\log^2 n \log^2 \log n))$ times optimal [FM00].)

This result — and its incorporation of the concepts of metric space approximation, unfairness, and connections to machine learning — form the launching point of the dissertation. Understanding these concepts and their connection to the metrical task system problem is the goal of the first part of the thesis, Chapters 2–4.

In Chapter 2, we learn how any metric space can be approximated by what are called HST spaces, a recent result from Bartal [Bar96, Bar98]. We also view a generalized form of MTS, called the unfair MTS problem, that allows us to build recursive algorithms for HSTs. This analysis indicates what sort of guarantee we want from our unfair MTS algorithm. We immediately see that this guarantee implies a substantial first step toward the $polylog(n)$ result.

Chapter 3 explains the machine learning problem called *predicting from expert advice* [LW94, FS97]. This problem is closely related to the unfair MTS problem, as we demonstrate by taking an expert-advice algorithm Share and using it for the unfair MTS problem to get the bound desired from Chapter 2.

Chapter 4 picks up from Chapter 2 again, describing an alternative algorithm Odd-Exponent achieving this same bound, and showing how to use Odd-Exponent in a more complicated way to get the $polylog(n)$ ratio. We observe that the same techniques work with Share, although at the loss of an $O(\log^2 n)$ factor.

The second part of the dissertation, Chapters 5–7, extends the concepts for the polylogarithmic bound (especially the connection to machine learning) to get the other three main results of the thesis.

Combining on-line algorithms

Chapter 5 discusses a problem called *combining on-line algorithms on-line*, where we, as the on-line algorithm, have a number of on-line algorithms which we might follow, but changing our current on-line algorithm has a cost. This algorithms might, for example, incorporate a number of heuristics which do well on particular event sequences, in case the actual event sequence matches one of our heuristics. Using Experts results, we see how we can guarantee that our on-line combination algorithm can do almost as well as the best of several on-line algorithms whose performance we can see.

We also see how an on-line algorithm might do if it can only see the performance of its current heuristic. For example, this might happen in process migration: We can have a heuristic for each computer, telling the process to stay at that computer. But if the process can only read the load average at its current location, it sees only its current heuristic's performance. Even if it can see only its current selection, our on-line algorithm can guarantee that it does not pay much more than if it knew in advance which heuristic pays least.

Relating metrical task systems and expert advice

In Chapter 6, we extend Chapter 3 by looking at the converse direction — using unfair MTS algorithms for the expert advice problem. In particular, while Chapter 3 explains that some Experts algorithms also make

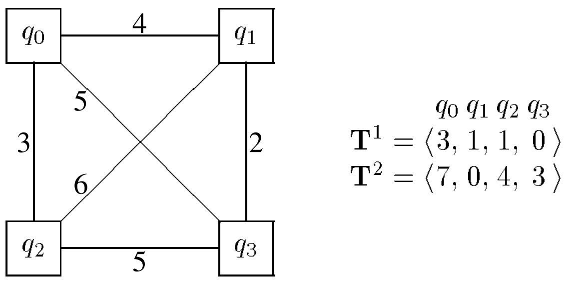


Figure 1.1: A metric space and task sequence.

good unfair MTS algorithms, Chapter 6 proves that *any* algorithm with an MTS guarantee implies a similar algorithm with an Experts guarantee.

To get a feel for the variety of algorithms this implies for Experts, we look at the results of a small experiment comparing how different MTS algorithms perform on a sample of process migration data.

The unfair paging problem

The final direction we take is to extend the notion of unfairness, which we employed in our analysis of the MTS problem, to paging. In particular, we compare the on-line algorithm's performance against the cost of servicing the request sequence if we increase the power of the off-line algorithm by allowing it to ignore the request at a cost of $1/r$. We see an on-line algorithm that guarantees that it pays no more than $O(r + \log k)$ times the best off-line cost computed with this added power. (Here k represents the cache size.)

In Chapter 7, we see the significance of the problem and how machine learning can be used to achieve improved results for it. Besides the significance of this problem to paging, this work can also be seen as a first effort at using the techniques used for the polylogarithmic guarantee for metrical task systems to achieve similar guarantees for the much more challenging k-server problem.

1.2 The metrical task system problem

The initial problem motivating this work, and a major focus of this thesis, is the **metrical task system** (MTS) problem due to Borodin, Linial, and Saks, designed to abstract a wide variety of on-line problems [BLS92].

Problem MTS ([BLS92]) We live in a system of n **states** with a distance metric d separating the states. This distance metric is nonnegative ($d(u, v) \geq 0$), is symmetric ($d(u, v) = d(v, u)$), and has the triangle inequality ($d(u, v) + d(v, w) \geq d(u, w)$). At all times we occupy a single state. At the beginning of each time step, we receive a **task vector**, specifying a nonnegative cost for each state (representing our cost if we process the task in that state). When we receive a task vector $\mathbf{T}$, we choose whether to stay at our current state or to move to a different state. We pay both for moving between states (according to d) and for processing the task (according to $\mathbf{T}$ at our new state). Our goal is to minimize our total cost over the task sequence.

Example 1.1 Consider the metric d and task sequence $\mathbf{T}$ illustrated in Figure 1.1. On $\mathbf{T}^1$,† we may choose to process the task in state q_2 and so pay $\mathbf{T}_2^1 = 1$ to process. Then say we choose to process

†This dissertation uses superscripts not only for exponentiation but also for indexing time. To relieve ambiguity, time-indexed variables appear in boldface.

$\mathbf{T}^2$ in state q_3. We pay $d(q_2, q_3) = 5$ to move and $\mathbf{T}^2_3 = 3$ to process the task. Our total cost on this sequence, then, is $1 + (5 + 3) = 9$. (We have chosen sub-optimally: The optimal choice is to start at q_1 and remain there, for a total cost of $1 + (0 + 0) = 1$.)

The importance of metrical task systems lies in the fact that they generalize many natural on-line problems. The following three examples illustrate this.

Example 1.2 Laptop computer power management inspires the following very simple task system. The states are q_0, representing that the laptop's hard drive is not spinning, and q_1, representing that it is. The distance between the states is half the amount of power required to begin spinning the disk. (We use half because to be a metric the distance function must be symmetric. We are optimizing on the total cost: Each time we move from q_0 to q_1, we will later move from q_1 to q_0; by using half each time, we add the full amount to the total.) On all time steps, the cost to q_1 is the amount of power to keep the disk spinning. For time steps where there is no disk access, the cost to q_0 is 0, but when there is a disk access, the cost to q_0 is infinite to prevent an on-line player from being in q_0 for the task. (Helmbold, Long, and Sherrod consider laptop disk management as a practical problem to be approached using machine learning theory [HLS96]. We relate machine learning theory to task systems in Chapter 3.)

Example 1.3 Say we have a computational process that can move on a network between computers with varying loads. In metrical task systems, the costs should represent the quantity we want to minimize, and in this case we want to avoid lost computation time. So the metric gives the lost time involved in transporting the process from one computer to another. And on each time step, the task vector tells us for each computer how much time would have been lost were we at that computer. (Section 6.3 describes an experiment comparing different MTS algorithms using computer load data.)

Example 1.4 Paging can be formulated in the metrical task system framework. If we have a cache that can hold k pages, and there are n pages in the universe, then the task system would include a state for each of the $\binom{n}{k}$ choices of k pages from the universe. Our current state tells us what we should hold in our cache. We represent a request to a page i as a task with a cost of 0 for those states where i is in the state's corresponding cache and ∞ elsewhere. The distance between two states is the number of page loads required to move between the two states' corresponding sets. (The MTS results in this thesis unfortunately say nothing useful about Paging, as the number of states is much too large to generate useful bounds. But Chapter 7 describes how the techniques used for the MTS results of this thesis can apply to Paging.)

Some definitions will help us discuss task systems. An **event sequence** (or **task sequence**) $\mathbf{T}$ is the time-indexed sequence of task vectors. An **action sequence** is a time-indexed sequence of states specifying where each task is processed; in Example 1.1, the action sequence $\mathbf{v}$ is $\langle q_2, q_3 \rangle$. The **movement cost** $move(\mathbf{v})$ is the total cost incurred according to the metric, $\sum_t d(\mathbf{v}^{t-1}, \mathbf{v}^t)$. The **local cost** (or **task-processing cost**) $local(\mathbf{T}, \mathbf{v})$ is the total cost incurred according to the task vectors, $\sum_t \mathbf{T}^t_{\mathbf{v}^t}$. Thus the total cost $cost(\mathbf{T}, \mathbf{v})$ for $\mathbf{v}$ on $\mathbf{T}$ is $move(\mathbf{v}) + local(\mathbf{T}, \mathbf{v})$.

1.3 Competitive ratio

In the MTS problem, as with other on-line problems, the **competitive ratio** proves a useful performance measure of an algorithm. Informally, this is the maximum, over all event sequences $\mathbf{T}$, of the ratio of the algorithm's cost on $\mathbf{T}$ against the best possible cost for servicing $\mathbf{T}$. In Example 1.1, this ratio is $9/1$. (But of course, since we looked at only one event sequence (and not all possible event sequences), this is not really a competitive ratio.) Sleator and Tarjan proposed this competitive ratio as a general technique for analyzing on-line algorithm performance [ST85a].

Example 1.5 A tourist visiting New York City for a day can pay $1.50 for a single subway trip and $4.00 for an all-day pass. A simple strategy employed by many tourists is to simply buy the $4.00 pass at the first subway ride, at a cost of $4.00. This has a poor competitive ratio, since if it is also the last ride, the ratio is $4/1.5 = 2.667$. An alternative strategy is to buy single-trip tokens for the first two rides and the all-day pass for the third. For this, the worst-case ratio is $7/4 = 1.75$, which occurs if the tourist takes three rides.

Example 1.5 illustrates that the competitive ratio is not always the most intuitive way of looking at the problem. If our tourist were quite sure she would use the subway more than twice, perhaps she should have bought the all-day pass initially. Or if our tourist brought only $5.00, she may want the all-day pass. The advantage of the competitive ratio bound is that it applies to many on-line problems without requiring additional input requirements (like a probability distribution) to the problem. Additionally, theoretical comparisons using competitive ratios often agree with empirical comparisons in how they rank algorithms. (Empirically, the ratios tend to be much lower since inputs generally are not adversarial).

Additional research refined the notion of competitive ratio slightly to incorporate randomization and to provide an additive fudge factor. We say randomized algorithm A is ρ-competitive if for any task sequence, the expected cost to A is at most ρ times the best achievable cost for the task sequence (plus a constant independent of the sequence). More formally, given a metric space d, an on-line algorithm A has competitive ratio ρ if for some constant b, for each event sequence $\mathbf{T}$, A outputs an action sequence $\mathbf{v}_A$ (a random variable if A is randomized) so that for all action sequences $\mathbf{v}$, the cost to A obeys the inequality

$$\mathbf{E}[cost(\mathbf{T}, \mathbf{v}_A)] \leq \rho \ cost(\mathbf{T}, \mathbf{v}) + b \ . \tag{1.1}$$

The **additive part** b proves to be an important (and irritating) detail. Thus we frequently speak of A as having "ratio ρ with additive b."

The way the quantifiers are ordered in this definition assumes an **oblivious adversary**; an adversary choosing the worst-case $\mathbf{T}$ must choose the entire sequence without knowing A's particular choices. This is appropriate in circumstances where the algorithm has a negligible effect on the environment — such as in paging (usually) and in small-quantity stock investing. An alternative is to use an **adaptive adversary** who can choose each task vector knowing A's random choices so far [BDBK+94]. But throughout this thesis we use an oblivious adversary for all our on-line problems.

One very nice aspect of analyzing algorithms against oblivious adversaries is the simplicity of expressing the cost in the uniform metric (where all interstate distances are 1). If $\mathbf{p}^{t-1}$ is our current probability distribution, and we move to distribution $\mathbf{p}^t$ in order to process the task $\mathbf{T}^t$, define $d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$ to be

$$\sum_{i:\mathbf{p}_i^{t-1} > \mathbf{p}_i^t} \left(\mathbf{p}_i^{t-1} - \mathbf{p}_i^t\right) = \sum_{i:\mathbf{p}_i^{t-1} < \mathbf{p}_i^t} \left(\mathbf{p}_i^t - \mathbf{p}_i^{t-1}\right) \ .$$

(Since both $\mathbf{p}^{t-1}$ and $\mathbf{p}^t$ are probability distributions and so sum to 1, these quantities are equal.) For the task $\mathbf{T}^t$, our expected cost is exactly $d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right) + \sum_i \mathbf{p}_i^t \mathbf{T}_i^t$. It is convenient to think of probability as a fluid being moved between states as time progresses, where the movement cost between time steps is the amount of fluid being transferred. Indeed, we can redefine the MTS problem as the on-line algorithm choosing a probability distribution, with the costs as just described, and so avoid the intricacies of probability altogether.

Theorem 1.1 *Say we are in the uniform metric. We can change our state probability distribution from $\mathbf{p}^{t-1}$ to $\mathbf{p}^t$ at an expected cost of $d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$.*

Proof. If the probability we are at our actual current state i should increase (i.e., $\mathbf{p}_i^{t-1} \leq \mathbf{p}_i^t$), then we do not move. But if that probability decreases ($\mathbf{p}_i^{t-1} > \mathbf{p}_i^t$), then we remain at i with probability $\mathbf{p}_i^t/\mathbf{p}_i^{t-1}$ and otherwise choose randomly from among the states j whose probabilities increase, choosing with probabilities $\left(\mathbf{p}_j^t - \mathbf{p}_j^{t-1}\right)/d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$.

The new probability distribution with this strategy is $\mathbf{p}^t$. For decreasing-probability states i, the probability we are there is the product of the chance we were already there ($\mathbf{p}_i^{t-1}$) and the chance we remain there given we were already there ($\mathbf{p}_i^t/\mathbf{p}_i^{t-1}$), and this product is $\mathbf{p}_i^t$. There is no chance that we move to i. For increasing-probability states i, we are there if we move to i or if we were at i already. The probability we move there from a decreasing-probability state j is the product of the chance we were at j (which is $\mathbf{p}_j^{t-1}$), the chance we move from j given we were there (which is $\left(\mathbf{p}_j^{t-1} - \mathbf{p}_j^t\right)/\mathbf{p}_j^{t-1}$), and the chance we move to i given that we are moving from j (which is $\left(\mathbf{p}_i^t - \mathbf{p}_i^{t-1}\right)/d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$). This product is $\left(\mathbf{p}_j^{t-1} - \mathbf{p}_j^t\right)\left(\mathbf{p}_i^t - \mathbf{p}_i^{t-1}\right)/d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$. Summing over all such j gives us $\mathbf{p}_i^t - \mathbf{p}_i^{t-1}$. We could also have already been at state i (and remained there) with probability $\mathbf{p}_i^{t-1}$, for a total probability of $\mathbf{p}_i^t$.

To get the total probability we move, we sum the chances that we move to each state. For decreasing-probability states, this chance is 0. For increasing probability states i, we have already seen that the chance we move there is $\mathbf{p}_i^t - \mathbf{p}_i^{t-1}$. Summing over all states gives us $d\left(\mathbf{p}^{t-1}, \mathbf{p}^t\right)$. ∎

A major open problem in competitive analysis is, "How small a competitive ratio can one guarantee for metrical task systems on arbitrary distance metrics?" A primary goal of this dissertation is to present a substantially improved answer to this question.

1.4 Previous results

Uniform metric

The simplest, most important, and best-understood metric for task systems is the **uniform metric**, where $d(u, v) = 1$ when $u \neq v$ (and $d(u, u) = 0$ for all u).

The Marking algorithm of Borodin, Linial, and Saks is a simple and useful algorithm for the uniform metric [BLS92]. (This algorithm is similar to the Marking algorithm used for Paging, which we review in Chapter 7 [FKL+91].)

Algorithm Marking ([BLS92]) The algorithm proceeds in phases. At the beginning of each phase all states are unmarked, and Marking chooses a uniform-random state to occupy. As tasks are received, Marking increases counters on each state, keeping track of the total processing cost for the state in this phase. (This counter will increase when the state incurs a cost, whether or not the algorithm occupies it.) When a state's counter reaches 1, we say that this state is *marked*. When the current state becomes marked, the algorithm moves to a random unmarked state. When all states are marked, Marking resets all marks and counters and begins a new phase.

Example 1.6 Consider 3 states q_0, q_1, and q_2, where Marking begins at q_0, with the task sequence

$$\begin{array}{rl} & \;\; q_0 \;\;\; q_1 \;\;\; q_2 \\ \mathbf{T}^1 = & \langle 0.5, 0.2, 0.0 \rangle \\ \mathbf{T}^2 = & \langle 0.2, 0.3, 2.0 \rangle \\ \mathbf{T}^3 = & \langle 0.0, 1.0, 1.0 \rangle \\ \mathbf{T}^4 = & \langle 1.0, 0.0, 0.0 \rangle \end{array}$$

Marking initially chooses a random state — say it chooses q_1 and so pays 0.2 for $\mathbf{T}^1$. The counters are now $\langle 0.5, 0.2, 0\rangle$. On $\mathbf{T}^2$, the counters become $\langle 0.7, 0.5, 2\rangle$; q_2 is now marked, but Marking is at q_1 and so remains there, at a cost of 0.3. On $\mathbf{T}^3$, the counters become $\langle 0.7, 1.5, 3\rangle$. The current state q_1 is now marked; the algorithm chooses randomly from the unmarked states $\{q_0\}$, so Marking must choose q_0, at a cost of $1 + 0$. On $\mathbf{T}^4$, all states become marked; Marking clears all counters and chooses a random state, say q_2. The cost is $1 + 0$; Marking's total cost for these four tasks is $0.2 + (0 + 0.3) + (1 + 0) + (1 + 0) = 2.5$.

The following theorem bounds the competitive ratio of Marking. Achlioptas, Chrobak, and Noga demonstrate the best possible bound for Marking, $2H_n - 1$ [ACN96].‡

Theorem 1.2 ([BLS92]) *Marking has competitive ratio $2H_n$ for uniform metric spaces.*

Proof. We analyze by phases. Any action sequence taken by an off-line algorithm must pay at least 1 in each phase (either 1 to move or 1 if it stays in the same state); we argue that Marking's expected cost is at most $2H_n$. Consider the first state to become marked. The probability that Marking ever goes to this state during the phase is $\frac{1}{n}$, and if so then Marking pays at most 2 for this state (at most 1 to move there, and at most 1 in local costs). Thus the expected cost to Marking at this state is at most $\frac{2}{n}$. Now consider the second state to become marked. The probability that Marking ever goes to this state is at most $\frac{1}{n-1}$, and if so then Marking pays at most 2 for this; thus the expected cost to Marking at this state is at most $\frac{2}{n-1}$. Generally, at the ith state to become marked in the phase, Marking expects to pay at most $\frac{2}{n-i+1}$ at that state. We sum over all states to get $2H_n$. ∎

On the other side, we know that no algorithm can guarantee a competitive ratio of less than H_n . Irani and Seiden nearly match this lower bound with an algorithm achieving the ratio $H_n + O(\sqrt{\log n})$ [IS98].

Theorem 1.3 ([BLS92]) *Every on-line algorithm for the uniform metric has a competitive ratio of at least H_n.*

Proof. Consider the following sequence constructed by an adversary who maintains the probability distribution on states used by the on-line algorithm A. The sequence proceeds in phases. The first task vector of the phase is 0 on all but the most-probable state q_1, where it is infinite. Since A is at q_1 with probability at least $\frac{1}{n}$, and it will pay 1 to move from q_1 to avoid the infinite cost, the expected cost to A is at least $\frac{1}{n}$. The second task vector is 0 everywhere except for q_1 and the most-probable state q_2. The expected cost on this task is at least $\frac{1}{n-1}$. We continue this until we reach $n - 1$ tasks, each time using task vectors that are 0 everywhere except at $q_1, \ldots, q_{i-1}$ and the most-probable state q_i. The total cost to A after these tasks is at least $H_n - 1$. For the final task vector of this phase, we give a cost of 1 to the remaining state q_n and 0 elsewhere; since A must be at q_n, the cost is 1 for a total cost of at least H_n to A.

An off-line algorithm knowing the sequence would be at q_n for the first $n - 1$ tasks, at no cost; on the nth task, it would move to the next phase's q_n, at a cost of 1. Since the algorithm can repeat these phases indefinitely, the competitive ratio of A is at least H_n. ∎

General metrics

The situation for arbitrary metrics is more challenging. In the metric space of Figure 1.2, for example, Marking does very poorly — it will likely pay at least 100 in most phases. A more promising alternative for metric spaces like that of Figure 1.2 is to merge q_0 and q_1 somehow and to combine this q_0–q_1 combination with q_2 using some algorithm like Marking — that is, to use Marking to combine q_0 and q_1 in isolation,

‡By H_n, we mean the nth harmonic number, $\sum_{i=1}^{n} \frac{1}{i}$.

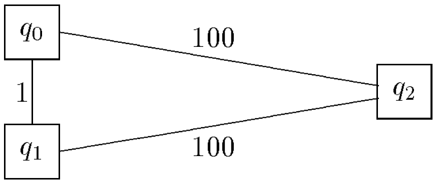


Figure 1.2: A decidedly nonuniform metric space.

and then to use Marking again to incorporate q_2 into the mixture. Karlin *et al.* consider the case of such an unbalanced 3-point space [KMMO90]; for larger unbalanced spaces, Blum *et al.* apply this principle of building from algorithms for subspaces [BKRS92]. This decomposition of a space into subspaces is also the inspiration behind the approach followed in this dissertation.

Many of the known algorithms, including many seen in this dissertation, use the **work function**. The work function $\mathbf{OPT}_v^t$, indexed by a time t and a state v, represents the optimal off-line cost for servicing the first t tasks and ending in state v. We can compute $\mathbf{OPT}_v^t$ as follows. Initially $\mathbf{OPT}_v^0$ is 0 for all v. Given a task vector $\mathbf{T}^t$ we update each state's work function to

$$\mathbf{OPT}_v^t = \min_u \left(\mathbf{OPT}_u^{t-1} + \mathbf{T}_u^t + d(u,v) \right) .$$

Notice that $\mathbf{OPT}_u$ and $\mathbf{OPT}_v$ can never differ by more than $d(u,v)$. We say that state u **pins** state v when $\mathbf{OPT}_v^t = \mathbf{OPT}_u^t + d(u,v)$.

Besides introducing the problem and presenting Marking, Borodin, Linial, and Saks also demonstrate a deterministic algorithm for general metric spaces.

Algorithm Work-Function ([BLS92]) We maintain the work function. When the state we occupy becomes pinned, we move to the pinning state.

Example 1.7 We return to Example 1.1. The work function values are initially $\mathbf{OPT}^0 = \langle 0,0,0,0 \rangle$. We initially occupy state q_0, and receive $\mathbf{T}^1 = \langle 3,1,1,0 \rangle$. We update our work function values to $\mathbf{OPT}^1 = \langle 3,1,1,0 \rangle$. Nobody yet pins state q_0, so we remain there, at a cost of 0 to move and 3 to process. Our second task vector $\mathbf{T}^2$ is $\langle 7,0,4,3 \rangle$, so our work function values become $\mathbf{OPT}^2 = \langle 5,1,5,3 \rangle$. Now state q_1 pins states q_0 and q_2. We are at state q_0, so we move to the pinning state, q_1, at a cost of 4 to move and 0 to process. Our total cost on this sequence, then, is $3 + (4 + 0) = 7$.

Borodin, Linial, and Saks show the following, not proven in this thesis.

Theorem 1.4 ([BLS92]) *Work-Function has competitive ratio $2n-1$ for any metric space.*

They complement this by showing that *deterministic* algorithms cannot guarantee less than $2n-1$.

How much better can one do with randomized algorithms? This remains a major open question in competitive analysis. It was not even clear that *any* improvement was possible until Irani and Seiden demonstrated a randomized algorithm with a mildly improved competitive ratio, $1.58n - 0.58$ [IS98]. On the lower-bound front, Blum *et al.* show that regardless of the metric, every algorithm must have a competitive ratio of at least $\Omega(\sqrt{\log n / \log\log n})$ [BKRS92].

In the absence of any satisfying bounds closing this gap for arbitrary metrics, researchers developed algorithms for some natural metrics beyond the uniform metric. These include an $O(\log n)$ ratio for "highly unbalanced spaces" [BKRS92], an $O(\log^2 n)$ ratio for a star space [Tom97], and a $2^{O(\sqrt{\log n \log\log n})}$ ratio for equally-spaced points on a line [BBF+90, BRS97]. (In a star space, $d(u,v)$ is $d_u + d_v$ for some choices d_v of values for states.)

These examples in other metrics led to the somewhat daring conjecture that a general algorithm exists achieving $O(\log n)$ on every metric, and that no metric exists where $o(\log n)$ is possible. This $O(\log n)$ algorithm remains elusive, but an algorithm, presented in this dissertation, achieves ratio $O(\log^5 n \log\log n)$. Fiat and Mendel subsequently refine this to $O(\log^2 n \log^2 \log n)$ [FM00]. These polylogarithmic guarantees, coupled with the $\Omega(\sqrt{\log n / \log\log n})$ lower-bound result of Blum *et al.* [BKRS92], gives strong evidence for the randomized MTS conjecture.

Chapter 2

HST approximation

Bartal's probabilistic approximation of arbitrary metric spaces with h-HSTs is a major new tool in optimization algorithm research [Bar96, Bar98]. The MTS problem was a major motivation behind this result, and the MTS result presented in this dissertation remains an important application. In this chapter we explore this result and its application to the MTS problem.

2.1 Probabilistic approximation

The notion of probabilistic approximation dates from Karp [Kar90]. A metric space d is **probabilistically approximated** with ratio ρ by a class $\widetilde{\mathcal{C}}$ of metric spaces with an associated distribution if, for every pair of points u and v in d,

1. For all metrics $\tilde{d} \in \widetilde{\mathcal{C}}$, we have $\tilde{d}(u, v) \geq d(u, v)$.
2. $\mathbf{E}_{\tilde{d} \in \widetilde{\mathcal{C}}}\left[\tilde{d}(u, v)\right] \leq \rho \cdot d(u, v)$.

That is, every edge expands (regardless of our choice of $\tilde{d}$, no edge becomes shorter than in d) but its expected expansion factor is not more than ρ.

Example 2.1 Karp uses a simple example of probabilistically 2-approximating an n-node cycle space by a set of n-node line spaces: Choose a random edge of the cycle and split it there. (See Figure 2.1.) No matter which edge we pick, no distance shrinks using this approximation. But for any adjacent pair of nodes u and v, the edge connecting them is split with probability $\frac{1}{n}$; otherwise

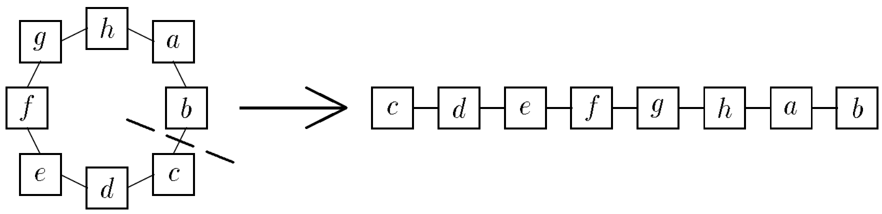


Figure 2.1: Approximating a cycle by a line.

it remains intact. Thus the expected distance is

$$\mathbf{E}_{\tilde{d}}\left[\tilde{d}(u,v)\right] \le \left(1-\frac{1}{n}\right)1+\frac{1}{n}(n-1) = 2-\frac{2}{n} \le 2\,.$$

For any nonadjacent pair of nodes, their expected distance in $\tilde{d}$ is at most the sum of the expected lengths along edges in the shortest path between them, and we know that these edges expand by $2-\frac{2}{n}$ in expectation.

The following straightforward theorem relates the concept of probabilistic approximation to the MTS problem. Coupled with Example 2.1, for example, it says that an MTS algorithm that is ρ-competitive on line spaces implies a 2ρ-competitive algorithm for cycle spaces.

Theorem 2.1 *Say that we can probabilistically ρ-approximate a metric space d with a distribution on a class $\widetilde{\mathcal{C}}$ of metric spaces, and say we can find an r-competitive MTS algorithm $\tilde{A}$ for metrics from $\widetilde{\mathcal{C}}$. Then we have an $(r\rho)$-competitive algorithm A for d.*

Proof. Our algorithm A probabilistically approximates d by a metric $\tilde{d} \in \widetilde{\mathcal{C}}$ and then runs $\tilde{A}$ on $\tilde{d}$ using the identical task sequence. On each step t, A chooses to occupy whichever state $\mathbf{v}^t_{\tilde{A}}$ that $\tilde{A}$ occupies within $\tilde{d}$.

Consider any action sequence $\mathbf{v}$ in d. Let $\mathbf{E}_{\tilde{d}}[\cdots]$ represent the expected cost of A relative to its choice of $\tilde{d}$, and let $\mathbf{E}_{\tilde{A}}[\cdots]$ represent the expected cost of $\tilde{A}$ given the choice of $\tilde{d}$. The expected cost to A is

$$\mathbf{E}_{\tilde{d}}\left[\mathbf{E}_{\tilde{A}}\left[\sum_t d(\mathbf{v}^{t-1}_{\tilde{A}},\mathbf{v}^t_{\tilde{A}})+\mathbf{T}^t_{\mathbf{v}^t_{\tilde{A}}}\right]\right] \quad\le\quad \mathbf{E}_{\tilde{d}}\left[\mathbf{E}_{\tilde{A}}\left[\sum_t \tilde{d}(\mathbf{v}^{t-1}_{\tilde{A}},\mathbf{v}^t_{\tilde{A}})+\mathbf{T}^t_{\mathbf{v}^t_{\tilde{A}}}\right]\right]\,.$$

(The inequality holds because $d(u,v) \le \tilde{d}(u,v)$ necessarily.) The amount inside $\mathbf{E}_{\tilde{d}}[\cdots]$ on the right is exactly the expected cost to $\tilde{A}$ on $\tilde{d}$. Using the fact that $\tilde{A}$ is r-competitive, we continue.

$$\begin{aligned}
\mathbf{E}_{\tilde{d}}\left[\mathbf{E}_{\tilde{A}}\left[\sum_t \tilde{d}(\mathbf{v}^{t-1}_{\tilde{A}},\mathbf{v}^t_{\tilde{A}})+\mathbf{T}^t_{\mathbf{v}^t_{\tilde{A}}}\right]\right] &\le \mathbf{E}_{\tilde{d}}\left[r\sum_t\left(\tilde{d}(\mathbf{v}^{t-1},\mathbf{v}^t)+\mathbf{T}^t_{\mathbf{v}^t}\right)+b\right]\\
&= r\sum_t\left(\mathbf{E}_{\tilde{d}}\left[\tilde{d}(\mathbf{v}^{t-1},\mathbf{v}^t)\right]+\mathbf{T}^t_{\mathbf{v}^t}\right)+b\\
&\le r\sum_t\left(\rho d(\mathbf{v}^{t-1},\mathbf{v}^t)+\mathbf{T}^t_{\mathbf{v}^t}\right)+b\\
&\le r\rho\sum_t\left(d(\mathbf{v}^{t-1},\mathbf{v}^t)+\mathbf{T}^t_{\mathbf{v}^t}\right)+b
\end{aligned}$$

Since this inequality holds for any sequence $\mathbf{v}$, A is $(r\rho)$-competitive. ∎

2.2 Approximation with HSTs

Bartal's contribution is to develop a technique for approximating arbitrary metrics by a special type of space particularly amenable to constructing algorithms, the *h-hierarchical well-separated tree* (h-HST). Define the **diameter** of a metric space to be the maximum distance separating any two points in it. A metric space with diameter Δ is an h-HST metric if it can be partitioned into subspaces that are recursively h-HST metrics with diameters at most Δ/h, where the distance between any two points in different subspaces is

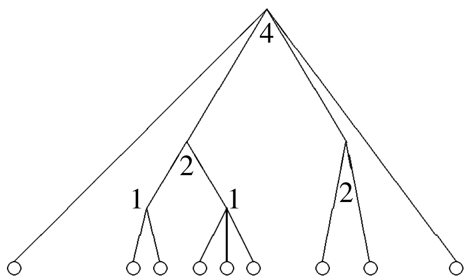


Figure 2.2: An example of a 2-HST. (The circles are points, and the numbers indicate diameters of subtrees.)

Δ.* The easiest way to draw an h-HST is as a tree; see Figure 2.2. In this drawing, the distance between the second point and fifth points from the left is 2, since this is the diameter of the lowest subtree containing both points.

Theorem 2.2 ([Bar98]) *For any $h \geq 1$, any metric space of n nodes can be probabilistically approximated with ratio $O(h \log n \log\log n)$ by a distribution on h-HSTs.*

Some of our less sophisticated results rely on the number of levels in the h-HST; in this theorem, the depth of each tree is $O(\log_h \Delta)$, where Δ is the ratio of the longest distance to the shortest nonzero distance in d.

This theorem has many applications to approximation algorithms and on-line algorithms. For many of these cases, the value of h is irrelevant and so h is taken to be simply 1. But in the MTS result we will find it necessary to take h to be a larger value (like $O(\log n)$).

Rather than look at the proof of Theorem 2.2, for intuition we look at a simplified result applying only to ℓ_∞ metrics, and then we briefly discuss how the same approach applies to arbitrary ℓ_p metrics. (In an ℓ_∞ space, points have coordinates, and the distance $d(u, v)$ between two points u and v is $\max_i |u_i - v_i|$, where u_i is the ith coordinate of point u.)

Theorem 2.3 *For any $h > 1$, any k-dimensional ℓ_∞ space of n nodes can be probabilistically approximated with ratio $O(hk \log_h n)$ by h-HSTs.*

Algorithm Approx-ℓ_∞ Say our metric space d has diameter D. We construct our h-HST by selecting, for each dimension, a partition of the axis into pieces of width $\frac{D}{h}$. Independently for each axis, we choose the offset of this partition by choosing a number uniformly from $[0, \frac{D}{h}]$ so that no pair of nodes $u, v \in d$ with $d(u, v) < \frac{D}{n^2 h}$ is divided. (That is, we continue choosing new offsets until no such pair is split by our choice. Finding such a partition is always possible; there are at most $\frac{n^2}{2}$ pairs of points, so at most $\frac{n^2}{2} \frac{D}{n^2 h} = \frac{D}{2h}$ of the range $[0, \frac{D}{h}]$ is disallowed.) This produces a partition of the k-dimensional space into at most $(h+1)^k$ nonempty regions, which we call **divisions**. Our h-HST will have a recursively-computed subspace for each division. We choose the diameter (that is, the distance between points in different divisions) to be D. Because each division has diameter at most $\frac{D}{h}$ (and so the recursively-computed subspace has diameter at most $\frac{D}{h}$), we get an h-HST. Figure 2.3 illustrates this technique on a 2-dimensional ℓ_∞ space with $h = 2$.

Proof. Consider any pair of nodes u and v in our original space. This pair will be separated on some level of the tree; since the diameter D on that level is at least $d(u, v)$, we satisfy the first requirement

*Bartal's definition of the distance between two points u and v is different: Whereas we define it to be the diameter of the lowest subspace containing u and v, he defines it as the sum of this "diameter" and half the sum of the "diameters" of the subspaces in each lower level containing u or v [Bar96]. (Bartal's definition comes from mapping the space to a tree with lengths assigned to the edges and points at the leaves. The distance from u to v is the sum of edge lengths on the path from u to v in the tree.) Since we always use $h \geq 2$, the two definitions differ by only a constant factor.

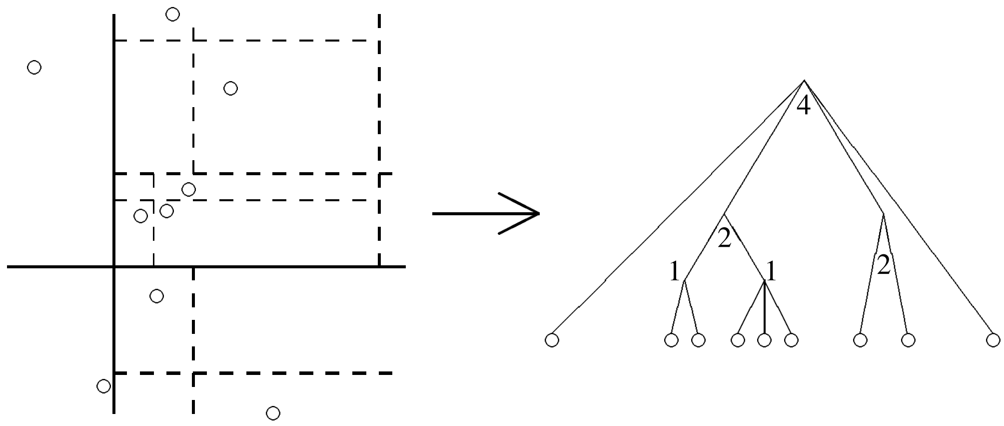


Figure 2.3: Constructing a 2-HST for an ℓ_∞ space. (Circles are points; on the left, distances are based on the two-dimensional coordinates in the diagram, and lines represent the partitions.)

of a probabilistic approximation, $d(u,v) \le D = \tilde{d}(u,v)$. Now we consider the upper bound on the expected $\tilde{d}(u,v)$. The nodes u and v will be split on a level of the recursion where the diameter is between $d(u,v)$ and $n^2 h d(u,v)$. There are at most $1 + \log_h(n^2 h) = O(\log_h n)$ of these. For a level of recursion with a diameter D, for each coordinate the probability that the partition splits u and v is at most $\frac{d(u,v)}{D/2h}$, and in this case $\tilde{d}(u,v)$ is D. So the expected contribution to the distance is at most $2hd(u,v)$. We sum over all coordinates to get $2hkd(u,v)$, and sum over all $O(\log_h n)$ levels to get

$$\mathbf{E}\left[\tilde{d}(u,v)\right] = O(hk \log_h n) d(u,v) .$$

■

This approach generalizes naturally to arbitrary ℓ_p metrics.

Theorem 2.4 *For any $h > 1$ and integer $p \ge 1$, any k-dimensional ℓ_p metric space of n nodes can be probabilistically $O(hk \log_h(nk^{1/p}))$-approximated by h-HSTs.*

Proof. We follow the method of Theorem 2.3, with a few differences. When the diameter is D, we partition each axis into pieces of width $\frac{D}{hk^{1/p}}$ so that the diameter of each division is $\frac{D}{h}$, but we choose the offset so that no point pair (u,v) with $d(u,v) \le \frac{D}{n^2 hk^{1/p}}$ is separated. Consider any pair of points u and v. For each coordinate i, let $\ell_i = |u_i - v_i|$. The chance the pair is split by the partition on coordinate i when the diameter is D is at most $\frac{\ell_i}{D/2hk^{1/p}}$. Summing over i, since (as shown below) $\sum_i \ell_i \le k^{1-1/p} d(u,v)$, we get at most a $2hkd(u,v)/D$ chance that $\tilde{d}(u,v) = D$. Thus the expected value of $\tilde{d}(u,v)$ is at most $O(hk \log_h(nk^{1/p}))d(u,v)$.

To show $\sum_i \ell_i \le k^{1-1/p} \left(\sum_i \ell_i^p\right)^{1/p}$, we show $\left(\sum_i \ell_i\right)^p \le k^{p-1} \sum_i \ell_i^p$ by induction on p. It trivially holds for $p = 1$. Given the fact for $p-1$, we have by induction

$$\begin{aligned} \left(\sum_i \ell_i\right)^p &\le k^{p-2} \left(\sum_i \ell_i^{p-1}\right) \left(\sum_i \ell_i\right) \\ &= k^{p-2} \sum_i \sum_j \ell_i^{p-1} \ell_j \\ &\le k^{p-1} \sum_i \ell_i^p \end{aligned}$$

The last step follows since $\ell_i^{p-1}\ell_j+\ell_j^{p-1}\ell_i \le \ell_i^p+\ell_j^p$ (this is equivalent to $(\ell_i^{p-1}-\ell_j^{p-1})(\ell_i-\ell_j) \ge 0$). ∎

2.3 Recursive MTS construction

Bartal's probabilistic approximation of general metrics by HSTs suggests a definite program for achieving improved probabilistic MTS algorithms: We find an algorithm for HSTs and apply Theorem 2.1. Because of their structure, a very natural approach for tackling HSTs is to inductively apply an algorithm. The $polylog(n)$ result described in this dissertation follows exactly this program.

A major hurdle is to conceive of a good scenario to abstract the details of algorithms for subtrees of an HST, so that we can define simple techniques to combine these into an algorithm for the entire tree using recursion. The remainder of this chapter describes this abstraction and demonstrates how to apply it.

To inductively construct our algorithm for the entire HST, we imagine that we already have r-competitive subalgorithms for each subtree of the root, and we construct an algorithm to combine these into an algorithm for the entire tree. We can abstract the r-competitiveness of the subalgorithms by imagining that each time the task vector says we pay δ, in fact our on-line algorithm pays $r\delta$. We will compare it to a player who does not incur this factor of r. We call this r the **cost ratio**; typically $r = polylog(n)$.

A complication that arises is that different subtrees can have different cost ratios. For the moment, though, we concentrate on the much simpler problem of finding an algorithm when the cost ratios are equal.

In using cost ratios, we speak of **unfair competitiveness**, a notion introduced by Blum *et al.* and formalized by Seiden [BKRS92, Sei99]. We say algorithm A has r-unfair competitive ratio ρ with additive b if for all event sequences $\mathbf{T}$, algorithm A outputs an action sequence $\mathbf{v}_A$ so that for all action sequences $\mathbf{v}$,

$$\mathbf{E}[move(\mathbf{v}_A) + r\ local(\mathbf{T}, \mathbf{v}_A)] \le \rho(move(\mathbf{v}) + local(\mathbf{T}, \mathbf{v})) + b\ . \tag{2.1}$$

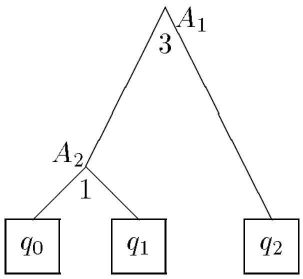
The only difference between this definition and the definition of the competitive ratio is the appearance of r on the left-hand side.

The first approach to consider, as Bartal did, is to analyze Marking in this unfair setting [Bar96].

Theorem 2.5 *Marking has r-unfair competitive ratio $(r + 1)H_n$ for a uniform metric space of n nodes.*

Proof. We analyze by phases. Any action sequence must pay at least 1 in each phase; we argue that Marking's expected unfair cost is at most $(r + 1)H_n$. Consider the first state to become marked. The probability that Marking ever goes to this state is $\frac{1}{n}$, and if so then Marking pays at most $r + 1$ for this state (at most r in local costs, and 1 in movement cost after it becomes marked). Thus the expected cost to Marking at this state is at most $\frac{r+1}{n}$. Now consider the second state to become marked. The probability that Marking ever goes to this state is $\frac{1}{n-1}$, and if so then Marking pays at most $r + 1$ for this; thus the expected cost to Marking at this state is $\frac{r+1}{n-1}$. Generally, at the ith state to become marked in the phase, Marking expects to pay at most $\frac{r+1}{n-i+1}$ at that state. We sum over all states to get $(r + 1)H_n$. ∎

It is not too difficult to imagine what happens when we apply Marking recursively to a tree. Because of the rH_n term to the competitive ratio, what effectively happens is that the H_n terms multiply so that for an L-level h-HST, the competitive ratio is roughly $O(H_n^L)$. The 1-level subtrees have ratio $O(H_n)$, but to construct the algorithm for the 2-level subtrees, we must take $r = O(H_n)$ to account for the performance of the 1-level subtrees below, giving a ratio of $O(H_n^2)$ overall. Likewise, the 3-level subtrees have a ratio of $O(H_n^3)$, and so on. We have neglected some details (notably, we have ignored details about exactly how



$$\begin{array}{rl} & \;\; q_0 \;\; q_1 \;\; q_2 \\ \mathbf{T}^1 = & \langle \tfrac{1}{2}, 0, 0 \rangle \\ \mathbf{T}^2 = & \langle \tfrac{1}{2}, \tfrac{1}{2}, 0 \rangle \\ \mathbf{T}^3 = & \langle 3, 3, 2 \rangle \\ \mathbf{T}^4 = & \langle 0, 0, 3 \rangle \end{array}$$

Figure 2.4: A very simple 2-HST and task sequence.

we combine the subalgorithms, and we have ignored the additive b), but this is roughly what happens in recursively applying Marking to an HST.

By choosing h to balance the metric-space approximation ratio h against the number of levels $O(\log_h \Delta)$, Bartal proves the following theorem.

Theorem 2.6 ([Bar96]) *Given a metric space with Δ as the ratio of longest to shortest distance, we choose $h = 2^{\sqrt{\lg \Delta \lg H_n}}$. By recursively applying Marking to an h-HST probabilistically approximating the original metric space, we get a competitive ratio of $2^{O(\sqrt{\log \Delta \log\log n})}$.*

In many cases (such as a shortest-path metric in an unweighted graph) this bound is an improvement on the earlier linear bounds [BLS92, IS98], but it is still much worse than the conjectured $O(\log n)$ possibility.

2.4 Bounding a competitive ratio

The key problem with the Marking approach is that Marking's unfair competitive ratio multiplies the ratio r by $2H_n = O(\log n)$. A ratio of $r + O(\log n)$ would be much more useful, as we could potentially add merely $O(\log n)$ for each level of the HST. In this section, we see how we can rigorously use such an algorithm A with an r-unfair competitive ratio of $r + \alpha(n)$ to recursively construct an algorithm for an L-level h-HST with a (fair) competitive ratio of $L\alpha(n)$, for h sufficiently large.

The techniques used here are later reused with less description in the $\mathit{polylog}(n)$ result. For that result, we must work around the fact that an h-HST could have many levels. For example, the space defined by placing points at $1, 2, 4, \ldots, 2^{n-1}$ on the number line will give an HST of $\Omega(\log_h 2^{n-1})$ levels. It turns out, though, that by being more careful with how we combine subspaces if one is much larger than others, we can get the $\mathit{polylog}(n)$ result. We will see this approach in Theorem 4.8.

To run A recursively on an HST T with each point of the space representing a subtree of T, we must decide when a point representing a subtree incurs a task-processing cost. We accomplish this by maintaining the work function $\mathbf{OPT}$ for the points *in that subtree alone*. (That is, points in other subtrees cannot pin any points in the subtree.) The point representing a subtree incurs a loss each time the minimum work function within that subtree increases. The amount of the loss is scaled down by the diameter of T (technically, a little less) and fed into A.

As A progresses at the root level of the tree, it will occasionally move from one subtree to another. When this occurs, the overall algorithm continues running A at that level, but for the lower levels of the HST (which have now changed subtrees) A begins anew. Restarting the algorithm in this way does not affect the work-function computation for the level where the movement occurs, but the work-function computation at the lower levels does begin from scratch.

Example 2.2 To get a handle on the subtleties of this scheme, we consider an example. We work with running Marking recursively on the HST and task sequence of Figure 2.4. (The choice of

Marking is inappropriate: It does not have the required $r + \alpha(n)$ competitive ratio. But Marking suffices for this illustration.)

Initially algorithm A_1 chooses between the left subtree and right subtree with equal probability; say it chooses the left subtree. Then algorithm A_2 runs and chooses between its left subtree and right subtree equally; say it chooses the left, so that the algorithm for the HST is initially at node q_0.

On receiving $\mathbf{T}^1 = \langle \frac{1}{2}, 0, 0\rangle$, the status of A_1 does not change; although the work function for u increases by $\frac{1}{2}$, the minimum work function within the subtree rooted at A_2 is still 0. However, the work function for left subtree of A_2 has increased by $\frac{1}{2}$. Thus Marking at A_2 increases the counter for the left subtree by $\frac{1}{2}$ (we divide the increase by the diameter of the space A_2). Algorithm A_2 does not move from q_0, and so we remain at q_0 to process the first vector.

On receiving $\mathbf{T}^2 = \langle \frac{1}{2}, \frac{1}{2}, 0\rangle$, the work function for both subtrees of A_2 increases by $\frac{1}{2}$; thus now the counters for A_2 are at 1 and $\frac{1}{2}$. Now its left subtree (q_0) is marked, so A_2 will move to the right subtree (q_1). At the root level, the left subtree's minimum work function is now 1, and so A_1's left counter increases from 0 to $\frac{1}{6}$ (remember that we scale by the space's diameter); A_1 does not move. So the algorithm processes the second vector at q_1.

For task $\mathbf{T}^3 = \langle 3, 3, 2\rangle$, the work function for A_1's left subtree increases by 3, so that A_1's left subtree counter increases from $\frac{1}{6}$ to $\frac{7}{6}$. Meanwhile, A_1's right subtree's work function increases by 2, so A_1's right subtree counter increases from 0 to $\frac{2}{3}$. Thus A_1's left subtree becomes marked, and it moves to the right subtree. The algorithm processes $\mathbf{T}^3$ at node q_2.

Finally, consider the task $\mathbf{T}^4 = \langle 0, 0, 3\rangle$. This increases the work function for A_1's right subtree by 3, so that A_1's right subtree counter becomes $\frac{5}{3}$. Now the right subtree of A_1 is marked, and so Marking resets the counters and begins at a random space. Say it randomly chooses the left subtree. Then A_2 begins anew with work function and counters at 0; say it chooses the left also. Then the algorithm processes $\mathbf{T}^4$ at node q_0.

In this example, we treated the tree as an entire entity. We now look at what A_1 saw. It saw the following task sequence.

$$\begin{aligned}\mathbf{T}_A^1 &= \langle 0, 0\rangle\\ \mathbf{T}_A^2 &= \langle \tfrac{1}{6}, 0\rangle\\ \mathbf{T}_A^3 &= \langle 1, \tfrac{2}{3}\rangle\\ \mathbf{T}_A^4 &= \langle 0, 1\rangle\end{aligned}$$

As a Marking algorithm using $r = 1$, A_1 is in either tree with equal probability for tasks $\mathbf{T}_A^1$ and $\mathbf{T}_A^2$. The left subtree becomes marked with $\mathbf{T}_A^3$, and so A_1 processes $\mathbf{T}_A^3$ in the right subtree. With $\mathbf{T}_A^4$, the right subtree becomes marked also, and so A_1 clears its marks and chooses a random subtree for $\mathbf{T}_A^4$.

To bound the performance of our recursive application, we must have a bound on the magnitude of the additive part (the b of our definition of competitive ratio in (2.1)). We need h to be about as large as b, so that when the subtree algorithm restarts, the additive part (which we may pay) will be only a constant factor more than it cost us to move into the subtree. We will see this in the mathematics of the formal proof.

Theorem 2.7 *Say algorithm A has r-unfair competitive ratio $r + \alpha(n)$ with additive $\beta(n) \geq 2$ on the uniform metric. The competitive ratio of running A recursively on an L-level $(2.5\beta(n))$-HST with diameter D is at most $1 + 4\alpha(n)L$ with additive $5\beta(n)D$.*

Remark. In running A, we take r to be $\frac{1}{4}$ times the maximum ratio of the subtrees' algorithms; $\beta(n)$ is computed using this value.

Proof. We prove this by induction on L. The trivial single-point space handles the base case $L = 0$.

Say we have an L-level HST of diameter D, and let ρ be the maximum competitive ratio of the subtrees' algorithms, at most $1 + 4\alpha(n)(L-1)$. The additive part is $5\beta(n)$ times the subtree's diameter of at most $\frac{D}{2.5\beta(n)}$, for a product of $2D$.

To bound the overall performance, we will want to use our inductive hypothesis and the r-unfair competitive ratio of A. To discuss A's performance, we define $\mathbf{T}_A$ as the task sequence that A sees. That is, $\mathbf{T}^t_{A,i}$ is $\frac{4}{3D}$ times the change in minimum work function in subtree i as a result of the actual task vector $\mathbf{T}^t$, where we compute the minimum work function in subtree i considering only those states in the subtree (i.e., in this computation, states in other subtrees cannot pin states in subtree i). We divide the change in work function by $\frac{3}{4}D$ rather than simply D because of the effect which will soon appear of the additive part of the subalgorithms' ratios.

To bound the competitive ratio for our complete algorithm (which combines A with the subtrees' algorithms), consider an arbitrary action sequence $\mathbf{v}$ on the entire space of n points. This implies an action sequence $\mathbf{v}_b$ specifying in which *subtree* (not state) to process each task. To use A's competitive ratio, we want to bound from below the total off-line cost to $\mathbf{v}$ in terms of $\mathit{local}(\mathbf{T}_A, \mathbf{v}_b)$ and $\mathit{move}_U(\mathbf{v}_b)$, since their sum is what A can compete against. (We use move_U to represent the movement cost on the diameter-1 uniform space that A uses.) The first apparent (but flawed) answer is $\mathit{local}(\frac{3}{4}D\mathbf{T}_A, \mathbf{v}_b) + D\ \mathit{move}_U(\mathbf{v}_b)$. To understand this, consider a segment of time where $\mathbf{v}$ stays within the same subtree. The algorithm must move into the subtree, at a cost of D. And because the work function within the subtree increases according to $\frac{3}{4}D\mathbf{T}_A$ within the segment, the off-line cost increases with $\frac{3}{4}D\mathbf{T}_A$. Summing over all segments, we get $\mathit{local}(\frac{3}{4}D\mathbf{T}_A, \mathbf{v}_b) + D\ \mathit{move}_U(\mathbf{v}_b)$.

But $\mathit{local}(\frac{3}{4}D\mathbf{T}_A, \mathbf{v}_b)$ is not accurate: The minimum cost for processing a segment of $\mathbf{v}$ remaining in the same subtree should be computed using work-function values starting at 0, but the work-function values used to compute $\mathbf{T}_A$ are not all equal (except for the first segment). In fact, for each of these $\mathit{move}_U(\mathbf{v}_b)$ segments, the actual optimal cost within the segment and the cost represented by $\mathbf{T}_A$ may differ by as much as the diameter of the subtree, which is at most $\frac{D}{2.5\beta(n)}$. So the first apparent answer $\mathit{local}(\frac{3}{4}D\mathbf{T}_A, \mathbf{v}_b) + D\ \mathit{move}_U(\mathbf{v}_b)$ may be wrong by as much as $\frac{D}{2.5\beta(n)}\mathit{move}_U(\mathbf{v}_b)$. Thus the total cost for $\mathbf{v}_b$ is at least

$$\mathit{local}(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_b) + \left(D - \frac{D}{2.5\beta(n)}\right) \mathit{move}_U(\mathbf{v}_b) \geq \mathit{local}(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_b) + \tfrac{3}{4}D\ \mathit{move}_U(\mathbf{v}_b)\ .$$

Now we look at what algorithm A does. Let $\mathbf{v}_A$ represent the sequence of moves that A makes at the top level of the HST. Within a single segment of $\mathbf{v}_A$ staying within a single subtree, the expected cost (according to the inductive hypothesis) is at most ρ times the optimal cost for servicing this segment, plus $2D$. Again, it is tempting to use $\mathbf{T}_A$ to bound the optimal cost for servicing the segment, but work-function discrepancies mean this estimate may be off: The proper way to compute the optimal cost is with the work function zero at all states at the beginning, while when the algorithm moves into the subtree, the work function varies between states. In this case, however, the perceived cost (that is, what $\mathbf{T}_A$ indicates) is at most the actual cost, since the computation using $\mathbf{T}_A$ only happens to believe that some of the states have incurred more cost than the minimum among the states, whereas in fact they have not. Thus within each of the $\mathit{move}_U(\mathbf{v}_A) + 1$ segments of $\mathbf{v}_A$, our expected cost is at most ρ times the local cost (according to the task sequence $\mathbf{T}_A$ that A sees) plus $2D$. Adding another D for each time we move between segments, our total cost is at most

$$\rho\ \mathit{local}\left(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_A\right) + 3D\ \mathit{move}_U(\mathbf{v}_A) + 2D\ .$$

Of course $\mathbf{v}_A$ is actually a random variable based on A's random choices. Since A has competitive ratio $r + \alpha(n)$, we know that for an arbitrary action sequence $\mathbf{v}_b$, A's expected cost is at most

$$
\begin{aligned}
&\mathbf{E}\left[\rho\ \mathit{local}\left(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_A\right) + 3D\ \mathit{move}_U(\mathbf{v}_A) + 2D\right] \\
&\quad = \quad 3D\,\mathbf{E}\left[\frac{\rho}{4}\ \mathit{local}\,(\mathbf{T}_A, \mathbf{v}_A) + \mathit{move}_U(\mathbf{v}_A)\right] + 2D \\
&\quad \leq \quad 3D\ \left(\left(\frac{\rho}{4} + \alpha(n)\right)(\mathit{local}\,(\mathbf{T}_A, \mathbf{v}_b) + \mathit{move}_U(\mathbf{v}_A)) + \beta(n)\right) + 2D \\
&\quad = \quad (\rho + 4\alpha(n))\left(\mathit{local}\left(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_b\right) + \tfrac{3}{4}D\ \mathit{move}_U(\mathbf{v}_A)\right) + 3\beta(n)D + 2D \\
&\quad \leq \quad (1 + 4\alpha(n)L)\left(\mathit{local}\left(\tfrac{3}{4}D\mathbf{T}_A, \mathbf{v}_b\right) + \tfrac{3}{4}D\ \mathit{move}_U(\mathbf{v}_A)\right) + 5\beta(n)D \\
&\quad \leq \quad (1 + 4\alpha(n)L)\,(\mathit{local}(\mathbf{T}, \mathbf{v}) + \mathit{move}(\mathbf{v})) + 5\beta(n)D
\end{aligned}
$$

Thus we conclude that our overall competitive ratio for the HST is $1 + 4\alpha(n)L$, plus an additive $5\beta(n)D$. ■

Our goal now is to demonstrate an algorithm with an r-unfair competitive ratio of $r + O(\log n)$. One way to this goal is to detour into machine learning theory. We pursue this now.

Chapter 3

The expert prediction problem

As the MTS problem is foundational to competitive analysis, so the problem of prediction from expert advice is foundational to on-line machine learning theory. It has several specific formulations. In this chapter we first look at one of the more traditional formulations, Experts-Predict, and then we examine more closely a "decision-theoretic" formulation. From there we can derive new analyses of algorithms in the decision-theoretic formulation that do well with a particular goal called *partitioning bounds*, and we can attempt to translate these bounds to the r-unfair MTS problem.

3.1 Classical formulation

Littlestone and Warmuth proposed the initial Experts-Predict problem.

> **Problem Experts-Predict ([LW94])** We see a set of n **experts**. For each time step, each expert makes a Boolean prediction. We decide on a Boolean prediction, and then we learn the correct answer. Our goal is to minimize the number of mistakes we make relative to the most accurate expert.*

For example, we might think of the experts as meteorologists predicting whether it will rain tomorrow. We want to predict well relative to the most talented among them without too many mistakes along the way.

From a learning perspective, this question models a situation where we have a set of hypotheses (termed *experts*), one of which predicts fairly accurately how the world operates. The question is how quickly we can converge on a good predictor. Thus, our goal is to bound how much worse we do relative to the best single expert.

The **mistake bound** of an algorithm bounds the number of mistakes the algorithm makes [Lit88]. In contrast to much of machine learning, mistake bounds do not employ distributional assumptions. That is, the experts need not perform uniformly over time in any sense. Despite the absence of such assumptions, the theoretical bounds obtained are surprisingly good.

*The algorithms actually extend to bounded real-valued predictions, with a loss function (such as square loss or log loss) assigning the penalties. With the square loss function, for example, if an expert predicts x and the true answer is y, the loss is $(x-y)^2$.

If one of the experts predicts perfectly, then the Halving algorithm does optimally for deterministic algorithms.

Algorithm Halving ([Mit82]) We keep track of a set P of experts, initially including all of them. Each time step, we predict whatever the majority of experts in P predict. Once we receive the true answer, we remove from P all experts who predicted wrong.

Obviously, each time Halving predicts wrong, the size of P goes down by at least half. Thus the mistake bound of Halving is $\lfloor \lg n \rfloor$.

When none of the experts predict perfectly, the problem becomes harder. One simple approach (as we saw with Marking) is to proceed in phases: In each phase, we run Halving until P becomes empty. If the best expert makes m mistakes, then this phased version of Halving makes at most $m \lfloor \lg n \rfloor$ mistakes.

Littlestone and Warmuth's weighted-majority algorithm WM does significantly better.

Algorithm WM ([LW94]) We use a parameter $\beta \in (0,1)$ and maintain a weight $\mathbf{w}_i$ with each expert, initially $\mathbf{w}_i^0 = 1$. At time step t, we predict according to a weighted majority of the experts, where each expert gets a weight of $\mathbf{w}_i^{t-1}$. Once we learn the correct answer, we update the weight of each expert who was mistaken to become $\mathbf{w}_i^t \leftarrow \mathbf{w}_i^{t-1}\beta$.

Example 3.1 Take $\beta = \frac{1}{3}$. Say we have four experts, x_0, x_1, x_2, and x_3. Our weights are initially $\mathbf{w}^0 = \langle 1, 1, 1, 1 \rangle$.

Say that x_0 predicts `false` on the first time step while the others predict `true`. Then we predict `true`, since it has weight 3 while `false` has weight 1. We then learn the true answer, `false` in this example. We update the weights to become $\mathbf{w}^1 = \langle 1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \rangle$.

Say that x_0 and x_1 predict `true` on the second time step, and x_2 and x_3 predict `false`. Then `true` has weight $\frac{4}{3}$ while `false` has $\frac{2}{3}$; our algorithm predicts `true`. If this is correct, then the weights are updated to become $\mathbf{w}^2 = \langle 1, \frac{1}{3}, \frac{1}{9}, \frac{1}{9} \rangle$.

On the third time step, if x_0 predicts `false` and the others predict `true`, then we predict `false`, since `false` has weight 1 and `true` has weight $\frac{5}{9}$.

The beauty of WM lies in the fact that, despite its simplicity, its bound is quite strong. The proof is cute; we repeat its technique several times in this chapter.

Theorem 3.1 ([LW94]) *For any expert k, WM has mistake bound*

$$m_{\mathsf{WM}} \leq \frac{1}{\ln\left(\frac{2}{1+\beta}\right)} \left(m_k \ln\left(\frac{1}{\beta}\right) + \ln n \right) ,$$

where m_k is the number of mistakes made by expert k.

Remark. To make better sense of this bound, let $\beta = 1 - 2\varepsilon$ for small ε. Then the bound translates to approximately $2(1+\varepsilon)m_k + \frac{1}{\varepsilon}\ln n$. Intuitively, this is an explicit trade-off between how quickly we settle on a particular expert (the $\frac{1}{\varepsilon}\ln n$ term) and how quickly we are able to adapt if that expert is actually bad but happens to do well for the first several rounds (the $2(1+\varepsilon)m_k$ term).

Proof. [LW94] Define $\mathbf{W}^t = \sum_i \mathbf{w}_i^t$ to be the total weight at time t and say m_{WM} is the number of mistakes WM makes. If WM makes a mistake at time t, then, since at least $\mathbf{W}^{t-1}/2$ weight is on the experts that err, the total weight decreases by at least $(1-\beta)(\mathbf{W}^{t-1}/2)$. Thus when WM makes a mistake, $\mathbf{W}^t$ is at most $\mathbf{W}^{t-1} - (1-\beta)(\mathbf{W}^{t-1}/2) = \frac{1+\beta}{2}\mathbf{W}^{t-1}$. Since WM makes m_{WM} mistakes, and since $\mathbf{W}^0 = n$, the final total weight $\mathbf{W}^{final}$ is at most $(\frac{1+\beta}{2})^{m_{WM}} n$.

On the other hand, $\mathbf{W}^{final}$ is at least the final weight $\mathbf{w}_k^{final}$ of expert k, which is exactly β^{m_k}. Thus we have

$$\beta^{m_k} \leq \mathbf{W}^{final} \leq \left(\frac{1+\beta}{2}\right)^{m_{WM}} n \ .$$

From here we take logarithms and solve for m_{WM} to get the result. ■

This bound is very close to twice the best expert's loss. Moreover, it says that we can double the number of experts (refining the hypothesis space by a factor of two) with very little increase in worst-case loss. A major strength of the theory of expert advice is how tight a bound we get with the very simple algorithm WM.

Both Halving and WM are deterministic. A randomized version of WM, which chooses experts randomly based on the weight distribution, roughly halves the bound on the expected loss to $(1+\varepsilon)m_k + \frac{1}{2\varepsilon}\ln n$ [LW94]. We see a proof of this in the Experts problem, an alternative formulation of Experts-Predict.

3.2 Decision-theoretic formulation

Freund and Schapire abstract away the aspect of combining expert predictions to arrive at what they term a "decision-theoretic" formulation of Experts-Predict [FS97]. We use this formulation throughout the remainder of this dissertation, so we refer to this problem simply as Experts.

Problem Experts ([FS97]) We see a set of n **experts**. For each time step t, we choose an expert $\mathbf{v}^t$. Then we learn the **loss vector**, $\boldsymbol{\ell}^t$, which specifies the loss $\boldsymbol{\ell}_i^t \in [0,1]$ of each expert for that time step. We incur the loss of the chosen expert, $\boldsymbol{\ell}_{\mathbf{v}^t}^t$. Our goal is to minimize the total loss we incur.

Any deterministic algorithm for Experts does at least n times worse than the best expert in the worst case. An adversary can construct a worst-case sequence by simulating the algorithm and each time step giving a loss of 1 to the expert that the algorithm will choose and a loss of 0 to the other experts. Thus after T time steps, the algorithm's cost is T, while the best expert's loss is at most T/n. Since $O(n)$ bounds are undesirable, we restrict our attention to randomized algorithms.

Given that one of the experts is perfect (that is, if for some i, at all times t we have $\boldsymbol{\ell}_i^t = 0$), we can use the following algorithm Rand-Halving, a randomized version of Halving and a degenerate instance of Hedge (discussed later). It has a loss of at most H_n.

Algorithm Rand-Halving Let P be a set of experts, initially including all experts. Each time step, we pick our an expert uniformly at random from P. Once we receive the loss vector, we remove from P all experts who incur some nonzero loss.

When all experts incur some loss, the problem becomes more complicated. The Hedge algorithm is Freund and Schapire's Experts adaptation of WM [FS97]. (In fact, the coefficients of Theorem 3.2's guarantee are optimal for on-line algorithms [Vov95, FS97].)

Algorithm Hedge ([FS97]) We use a parameter $\beta \in (0,1)$ and maintain a weight $\mathbf{w}_i$ with each expert, initially $\mathbf{w}_i^0 = 1$. At time step t, we choose expert i with probability proportional to its weight, $\mathbf{w}_i^{t-1}/\sum_j \mathbf{w}_j^{t-1}$. Given the loss vector, we update the weight of each expert to become $\mathbf{w}_i^t \leftarrow \mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}$.

Theorem 3.2 ([FS97]) *If an expert k incurs total loss $loss_k$, then Hedge incurs expected loss at most*

$$\mathbf{E}[loss_{\mathsf{Hedge}}] \leq \frac{\ln 1/\beta}{1-\beta} loss_k + \frac{1}{1-\beta}\ln n \ .$$

Remark. Again, to get a feel for the tradeoff, we make better sense of this bound by letting $\beta = 1 - 2\varepsilon$ for small ε. Then the mistake bound translates to approximately $(1+\varepsilon)loss_k + \frac{1}{2\varepsilon}\ln n$, roughly a factor of 2 less than WM's bound.

Proof. [FS97] Let $\mathbf{W}^t$ be the total weight $\sum_i \mathbf{w}_i^t$ at time t, and let $\mathbf{L}^t$ be the expected loss to Hedge at time t. Note that

$$\mathbf{L}^t = \sum_i \frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}} \boldsymbol{\ell}^t = \frac{1}{\mathbf{W}^{t-1}} \sum_i \mathbf{w}_i^{t-1} \boldsymbol{\ell}^t .$$

As in the proof of Theorem 3.1, we bound $\mathbf{W}^{final}$. We bound $\mathbf{W}^t$ in terms of $\mathbf{W}^{t-1}$.

$$\begin{aligned} \mathbf{W}^t &= \sum_i \mathbf{w}_i^t = \sum_i \mathbf{w}_i^{t-1} \beta^{\boldsymbol{\ell}_i^t} \\ &\leq \sum_i \mathbf{w}_i^{t-1} \left(1 - (1-\beta)\boldsymbol{\ell}_i^t\right) = \mathbf{W}^{t-1}\left(1 - (1-\beta)\mathbf{L}^t\right) \\ &\leq \mathbf{W}^{t-1} e^{-(1-\beta)\mathbf{L}^t} \end{aligned}$$

We can now bound $\mathbf{W}^{final}$.

$$\mathbf{W}^{final} \leq \mathbf{W}^{init} \prod_t e^{-(1-\beta)\mathbf{L}^t} = n e^{-(1-\beta)\sum_t \mathbf{L}^t} .$$

For the lower bound on $\mathbf{W}^{final}$, we know it is at least $\mathbf{w}_k^{final}$, which is exactly β^{loss_k}. Thus we have the inequality

$$\beta^{loss_k} \leq n e^{-(1-\beta)\sum_t \mathbf{L}^t} ,$$

which we solve for $\mathbf{E}[loss_{\mathsf{Hedge}}] = \sum_t \mathbf{L}^t$. ■

3.3 Partitioning bound

Until this point, we have contented ourselves with bounding performance against the best single expert over all time steps. The **partitioning bound** is a more ambitious goal. Here we try to do well against all partitions of time into intervals, where we pick the best expert within each time interval of the partition. Being able to do well against all partitions includes, for example, scenarios where one expert does very well for the first half of time, whereas another expert does best on the last half of time. For a good partitioning bound, an algorithm must adapt particularly quickly to changed expert performance.

Formally, given a partition P of time into intervals, let k_P be the number of intervals. We let $\mathbf{L}_P^j$ be the loss of the best expert within the jth interval, and we let L_P be the total loss over all intervals, $\sum_{j=1}^{k_P} \mathbf{L}_P^j$. The partitioning bound of algorithm A will be some bound on its expected loss of the form

$$\mathbf{E}[loss_A] \leq aL_P + bk_P$$

for some coefficients a and b.

We hope to find a generalized bound similar to Theorem 3.2's bound for Hedge, a bound of the form

$$\mathbf{E}[loss_A] \leq (1+\varepsilon)L_P + \tfrac{1}{\varepsilon} k_P \ln n .$$

We examine two variants of Hedge, Thresh and Share, that achieve this type of bound. In Section 7.2, we see another variant called Phased-Hedge.

Thresh

The first of these algorithms, Thresh, is an adaptation of Littlestone and Warmuth's WML algorithm to the Experts problem [LW94].

Algorithm Thresh We use parameters $\beta \in (0,1)$ and $\alpha \in (0, \frac{1}{2}]$, and maintain a weight $\mathbf{w}_i$ for each expert, initially $\mathbf{w}_i^0 = 1$. At time step t, we compute the total weight $\mathbf{W}^{t-1} = \sum_i \mathbf{w}_i^{t-1}$, and we let $\mathbf{S}^{t-1}$ be the set of experts i with $\mathbf{w}_i^{t-1} \geq \frac{\alpha}{n}\mathbf{W}^{t-1}$. Define $\widehat{\mathbf{W}}^{t-1}$ as the total weight in $\mathbf{S}^{t-1}$, $\sum_{i \in \mathbf{S}^{t-1}} \mathbf{w}_i^{t-1}$. We choose expert i with probability $\mathbf{w}_i^{t-1}/\widehat{\mathbf{W}}^{t-1}$ if $i \in \mathbf{S}^{t-1}$ and with probability 0 otherwise. Given the loss vector $\boldsymbol{\ell}^t$, for each expert $i \in \mathbf{S}^{t-1}$, we update its weight to become $\mathbf{w}_i^t \leftarrow \mathbf{w}_i^{t-1}\beta^{\ell_i^t}$; we do not change weights for $i \notin \mathbf{S}^{t-1}$.

Theorem 3.3 *Given n experts, Thresh incurs expected loss at most*

$$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) L_P + \left(\frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)}\right) k_P$$

for any partition P.

Remark. For small ε, let $\alpha = \frac{1}{n}$ and $\beta = 1 - 2\varepsilon$. As n becomes very large, the bound of Theorem 3.3 translates to approximately

$$(1+\varepsilon)L_P + \left(1 + \varepsilon + \tfrac{1}{\varepsilon}\ln n\right) k_P\ .$$

If we restrict our attention to $k_P = 1$ (the case considered in Theorem 3.2), we see that this effectively generalizes the bound of Hedge, at the loss of only a factor of 2 in the coefficient to $\ln n$.

Proof. [LW94] Note that $\widehat{\mathbf{W}}^t \geq (1-\alpha)\mathbf{W}^t$ for all t, and let $\mathbf{L}^t$ be the expected loss to Thresh at time t,

$$\mathbf{L}^t = \sum_{i \in \mathbf{S}^{t-1}} \frac{\mathbf{w}_i^{t-1}}{\widehat{\mathbf{W}}^{t-1}} \boldsymbol{\ell}_i^t\ .$$

As in Theorem 3.2's proof, we bound how a single step alters the total weight.

$$\begin{aligned}
\mathbf{W}^t &= \sum_{i \in \mathbf{S}^{t-1}} \beta^{\boldsymbol{\ell}_i^t}\mathbf{w}_i^{t-1} + \sum_{i \notin \mathbf{S}^{t-1}} \mathbf{w}_i^{t-1} \\
&\leq \sum_{i \in \mathbf{S}^{t-1}} (1-(1-\beta)\boldsymbol{\ell}_i^t)\mathbf{w}_i^{t-1} + \sum_{i \notin \mathbf{S}^{t-1}} \mathbf{w}_i^{t-1} \\
&= \mathbf{W}^{t-1}\left(1 - (1-\beta)\sum_{i \in \mathbf{S}^{t-1}} \frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}\boldsymbol{\ell}_i^t\right) \\
&\leq \mathbf{W}^{t-1}\left(1 - (1-\beta)(1-\alpha)\sum_{i \in \mathbf{S}^{t-1}} \frac{\mathbf{w}_i^{t-1}}{\widehat{\mathbf{W}}^{t-1}}\boldsymbol{\ell}_i^t\right) \\
&= \mathbf{W}^{t-1}\left(1 - (1-\beta)(1-\alpha)\mathbf{L}^t\right)
\end{aligned} \tag{3.1}$$

Consider any partition P, and examine segment j of the partition, where the best expert (call it k) incurs loss $\mathbf{L}_P^j$. Say that the total weight at the segment's beginning is $\mathbf{W}^{init}$ and the total weight at the segment's end is $\mathbf{W}^{final}$. Because Thresh never allows a weight to fall below $\beta\frac{\alpha}{n}\mathbf{W}^t$, the initial weight of expert k in the segment is at least $\beta\frac{\alpha}{n}\mathbf{W}^{init}$. Thus at the segment's end, expert k's weight, and hence $\mathbf{W}^{final}$, is at least $\beta^{\mathbf{L}_P^j}\beta\frac{\alpha}{n}\mathbf{W}^{init}$. Applying bound (3.1), we have

$$\beta^{\mathbf{L}_P^j}\beta\frac{\alpha}{n}\mathbf{W}^{init} \le \mathbf{W}^{final} \le \mathbf{W}^{init}\prod_t \left(1-(1-\beta)(1-\alpha)\mathbf{L}^t\right) .$$

So we have

$$\mathbf{L}_P^j \ln\beta + \ln\frac{\beta\alpha}{n} \le -(1-\beta)(1-\alpha)\sum_t \mathbf{L}^t ,$$

which gives us bound on the segment's expected loss of

$$\sum_t \mathbf{L}^t \le \frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\mathbf{L}_P^j + \frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)} .$$

Summing over segments, we get the desired bound. ■

Share

We also examine Share, an alternative to Thresh. This is an adaptation of Herbster and Warmuth's Variable-Share algorithm to the Experts environment [HW98].

Algorithm Share We use parameters $\beta \in (0,1)$ and $\alpha \in (0,\frac{1}{2}]$, and maintain a weight $\mathbf{w}_i$ for each expert, initially $\mathbf{w}_i^0 = 1$. At time step t, we choose expert i with probability proportional to its weight, $\mathbf{w}_i^{t-1}/\sum_j \mathbf{w}_j^{t-1}$. Given the loss vector, we update the weight of each expert to become $\mathbf{w}_i^t \leftarrow \mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t} + \frac{\alpha}{n}\boldsymbol{\Delta}^t$, where $\boldsymbol{\Delta}^t$ is $\sum_i \left(\mathbf{w}_i^{t-1} - \mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}\right)$.

The update rule used by this algorithm can be viewed as follows. We first update as usual: $\mathbf{w}_i^t \leftarrow \mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}$. This reduces the sum of the weights by some amount $\boldsymbol{\Delta}^t$. We then distribute an α fraction of this $\boldsymbol{\Delta}^t$ evenly among the n experts ($\frac{\alpha}{n}\boldsymbol{\Delta}^t$ each).

Theorem 3.4 *Given n experts, Share incurs expected loss at most*

$$\left(\frac{\ln(1/\beta)}{(1-\alpha)(1-\beta)}\right) L_P + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right) k_P$$

for any partition P.

Remark. For small ε, let $\alpha = \frac{1}{n}$ and $\beta = 1-2\varepsilon$. As n becomes very large, the bound of Theorem 3.4 translates to approximately

$$(1+\varepsilon)L_P + \tfrac{1}{\varepsilon}\ln n\, k_P .$$

That is, we get about the same tradeoff we saw with Hedge and Thresh.

Proof. Given a partition P, we consider segment i of the partition. Let $\mathbf{L}^t$ be the expected loss to Share at time step t within the segment. Say expert k is the best expert of the segment (with loss $\mathbf{L}_P^i$). Our goal is to show that the algorithm's expected loss $\sum_t \mathbf{L}^t$ is at most

$$\frac{\ln(1/\beta)\mathbf{L}_P^i + \ln(n/\alpha)}{(1-\beta)(1-\alpha)} \tag{3.2}$$

Such a bound, summed over segments, implies the theorem's bound.

Using the typical multiplicative-update analysis (Theorem 3.2) we get

$$\mathbf{W}^t \leq \mathbf{W}^{t-1}\left(1-(1-\beta)(1-\alpha)\mathbf{L}^t\right) .$$

So, if $\mathbf{W}^{init}$ is the sum of weights at the segment's beginning and $\mathbf{W}^{final}$ is the sum of weights at the segment's end, then $\mathbf{W}^{final}$ is bounded by

$$\mathbf{W}^{final} \leq \mathbf{W}^{init}\prod_t\left(1-(1-\beta)(1-\alpha)\mathbf{L}^t\right) . \tag{3.3}$$

Now consider the weight of expert k. At time t, we have $\mathbf{W}^t = \mathbf{W}^{t-1} - \mathbf{\Delta}^t + \alpha\mathbf{\Delta}^t$, and so $\mathbf{\Delta}^t$ is $\frac{1}{1-\alpha}(\mathbf{W}^{t-1}-\mathbf{W}^t)$. Thus the amount added to $\mathbf{w}_k^{t-1}$ due to the share update is $\frac{\alpha}{(1-\alpha)n}(\mathbf{W}^{t-1} - \mathbf{W}^t)$. In the entire segment, therefore, the total amount added to w_i due to the share updates is $\frac{\alpha}{(1-\alpha)n}(\mathbf{W}^{init}-\mathbf{W}^{final})$. Thus, even if $\mathbf{w}_k^{init}$ is zero, by the end of the segment we have

$$\mathbf{w}_k^{final} \geq \beta^{\mathbf{L}^i_P}\left(\frac{\alpha}{(1-\alpha)n}(\mathbf{W}^{init}-\mathbf{W}^{final})\right) , \tag{3.4}$$

since the worst case for $\mathbf{w}_k^{final}$ is if the penalties for the expert's losses come after the sharing.

For convenience, define

$$\Pi = \prod_t\left(1-(1-\beta)(1-\alpha)\mathbf{L}^t\right) .$$

Combining (3.3) and (3.4) we get

$$\beta^{\mathbf{L}^i_P}\frac{\alpha}{(1-\alpha)n}(\mathbf{W}^{init}-\mathbf{W}^{init}\Pi) \leq \beta^{\mathbf{L}^i_P}\frac{\alpha}{(1-\alpha)n}(\mathbf{W}^{init}-\mathbf{W}^{final}) \leq \mathbf{w}_k^{final} \leq \mathbf{W}^{final} \leq \mathbf{W}^{init}\Pi.$$

We can now solve for Π.

$$\Pi \geq \frac{\beta^{\mathbf{L}^i_P}\alpha}{(1-\alpha)n+\beta^{\mathbf{L}^i_P}\alpha} \geq \frac{\beta^{\mathbf{L}^i_P}\alpha}{n}$$

This gives us

$$-\ln\Pi \leq \ln\left(\frac{1}{\beta}\right)\mathbf{L}^i_P + \ln\left(\frac{n}{\alpha}\right) .$$

Recalling the definition of Π, we notice that

$$-\ln\Pi \geq (1-\beta)(1-\alpha)\sum_t \mathbf{L}^t ,$$

so

$$\sum_t \mathbf{L}^t \leq \frac{\ln(1/\beta)\mathbf{L}^i_P + \ln(n/\alpha)}{(1-\beta)(1-\alpha)}$$

as we desired in (3.2). ∎

3.4 Translating to MTS

The Experts and MTS problems have deep similarities: The experts correspond closely to MTS states, and the loss vectors correspond closely to task vectors. This gives us some hope that Thresh and Share can also be used as MTS algorithms. But there are some important differences between the problems.

- The MTS problem includes a cost for switching between states/experts.
- An MTS algorithm has *one-step lookahead*. That is, first the cost vector is announced, then the algorithm chooses whether to move, and finally the algorithm pays according to the entry in the cost vector for the new state. In contrast, the Experts algorithm has *zero lookahead*, in that it first pays and then moves.
- Because of the lookahead, MTS algorithms can deal with unbounded cost vectors. Large losses are actually advantageous to an on-line MTS algorithm in that they are essentially equivalent to allowing the algorithm to "see further into the future." That is, an adversary trying to defeat an MTS algorithm might as well use several small task vectors instead of a single large task vector, so that the algorithm is not sure which state is best. (Theorem 4.1 formalizes this observation.)
- The Experts goal of doing well with respect to the best expert is a much weaker goal than the competitive-ratio goal of doing well against all sequences. Of course, because the goal is weak, the Experts bounds are very good ($1 + \varepsilon$ times the best expert), whereas the MTS bounds are relatively poor ($O(\log n)$).

In this section we examine how our two Experts algorithms do in the unfair uniform-metric MTS problem. Later (Chapter 6) we look at the other direction — how MTS algorithms apply to the Experts scenario.

Thresh

Thresh, unfortunately, does not translate well in the unfair MTS setting. In fact, Thresh does not have a bounded ratio at all. Consider the two-expert case. Say that expert 2 incurs a loss large enough for its weight to drop to slightly below $\frac{\alpha}{2-\alpha}$. At this point, the algorithm has all probability on expert 1. Now suppose expert 1 incurs a tiny loss, just sufficient to bring w_2 to equal $\frac{\alpha}{n}W$. (Again, W stands for the total weight $\sum_i w_i$.) This forces the algorithm to move $\frac{\alpha}{2}$ probability over to expert 2. Now suppose expert 2 incurs an infinitesimal loss so that $w_2 < \frac{\alpha}{n}W$. This forces the algorithm to move $\frac{\alpha}{n}W$ probability back to expert 1. This situation can repeat indefinitely, causing the algorithm to incur unbounded movement cost with insignificant increase in the off-line optimal cost, giving an unbounded competitive ratio.

Share

The problem with Thresh is that it does not control its movement costs very smoothly. Share, however, does. In fact, we can show that it is good as a uniform-metric MTS algorithm. The bound for the MTS setting is exactly what we want from our discussion closing Section 2.3. (A new $\log r$ term appears, but this is not problematic since we can assume $r = O(n)$; if the ratio is higher, we can simply apply Work-Function to get the same guarantee.)

Theorem 3.5 *We use Share for the r-unfair uniform-metric MTS setting as follows: Given a task vector $\mathbf{T}^t$, we give $r\mathbf{T}^t$ to Share and use the resulting probability distribution to choose a state. Given any $\gamma \geq 2$, we can configure α and β in Share so that its r-unfair competitive ratio is*

$$\rho = r + 3.2\gamma \ln(n(r+1)) + 4$$

with an additive $\frac{\ell}{\gamma}$.

Remark. In the proof, we choose α to be $(r+1)^{-1}$. For β, we choose it to be $\left(1+\frac{\gamma}{r}\ln\frac{n}{\alpha}\right)^{-1}$ if $r \geq \gamma \ln(n(r+1))$ and $\frac{1}{e}$ otherwise.

Proof. Consider any off-line strategy $\mathbf{v}$. This corresponds to a partition $P_{\mathbf{v}}$ with $move(\mathbf{v})+1$ segments. The loss L_P of the partition is $local(\mathbf{T},\mathbf{v})$. We consider the local cost and the movement cost incurred by Share in turn. Theorem 3.4 shows that the task-processing cost satisfies

$$\begin{aligned}
&\mathbf{E}[r\, local(\mathbf{T},\mathbf{v}_A)] = \mathbf{E}[local(r\mathbf{T},\mathbf{v}_A)] \\
&\leq \left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) local(r\mathbf{T},\mathbf{v}) + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right)(1+move(\mathbf{v}))\,. \qquad (3.5)
\end{aligned}$$

(In fact, the MTS problem allows one-step lookahead; this only decreases the algorithm's cost.)

To analyze the movement cost, note that the total weight $\mathbf{W}^t$ only decreases with time. We show that for any time step t, the movement cost is at most $\ln(1/\beta)$ times the local cost.

$$\begin{aligned}
d(\mathbf{p}^{t-1},\mathbf{p}^t) &= \sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}} - \frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}+\frac{\alpha}{n}\boldsymbol{\Delta}^t}{\mathbf{W}^t}\right) \\
&\leq \sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}} - \frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^t}\right) \\
&\leq \sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}} - \frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^{t-1}}\right) \\
&\leq \sum_{i}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}} - \frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^{t-1}}\right) \\
&\leq \sum_{i}\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}\boldsymbol{\ell}_i \ln\left(\frac{1}{\beta}\right)
\end{aligned}$$

Thus the total r-unfair cost to Share is at most

$$\begin{aligned}
&\left(1+\ln\frac{1}{\beta}\right)\left(\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) r\, local(\mathbf{T},\mathbf{v}) + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right)(1+move(\mathbf{v}))\right) \\
&\leq \frac{1+\ln(1/\beta)}{(1-\beta)(1-\alpha)}\max\left\{r\ln\frac{1}{\beta},\ln\frac{n}{\alpha}\right\} cost(\mathbf{T},\mathbf{v}) + \frac{1+\ln(1/\beta)}{(1-\beta)(1-\alpha)}\ln\frac{n}{\alpha}\,.
\end{aligned}$$

We must choose the values of α and β appropriately.

If $r \geq \gamma\ln(n(r+1))$, we choose $\alpha = (r+1)^{-1}$ and $\beta = \left(1+\frac{\gamma}{r}\ln\frac{n}{\alpha}\right)^{-1}$. Since $\ln\frac{1}{\beta} \leq \frac{\gamma}{r}\ln\frac{n}{\alpha}$, and $(1-\beta)^{-1} = 1 + r/\left(\gamma\ln\frac{n}{\alpha}\right)$, the competitive ratio is

$$\begin{aligned}
&\frac{1+\ln(1/\beta)}{(1-\beta)(1-\alpha)}\max\left\{r\ln\frac{1}{\beta},\ln\frac{n}{\alpha}\right\} \\
&\leq \frac{1+\frac{\gamma}{r}\ln(n/\alpha)}{1-\alpha}\left(1+\frac{r}{\gamma\ln\frac{n}{\alpha}}\right)\max\left\{\gamma\ln\frac{n}{\alpha},\ln\frac{n}{\alpha}\right\} \\
&= \frac{1}{1-\alpha}\left(2+\frac{r}{\gamma\ln\frac{n}{\alpha}}+\frac{\gamma\ln\frac{n}{\alpha}}{r}\right)\gamma\ln\frac{n}{\alpha}\,.
\end{aligned}$$

We continue, using the fact that $r \geq \gamma \ln(n(r+1))$.

$$\begin{aligned}\frac{1}{1-\alpha}\left(2+\frac{r}{\gamma \ln \frac{n}{\alpha}}+\frac{\gamma \ln \frac{n}{\alpha}}{r}\right)\gamma \ln \frac{n}{\alpha} &\leq \left(1+\frac{1}{r}\right)(r+3\gamma \ln(n(r+1))) \\ &\leq r+3\gamma \ln(n(r+1))+4\end{aligned}$$

The additive part is identical to this derivation, except that $\max\{\cdots\}$ is replaced by $\ln \frac{n}{\alpha}$, a factor of γ less.

If $r < \gamma \ln(n(r+1))$, then we choose $\alpha = (r+1)^{-1}$ and $\beta = \frac{1}{e}$. The competitive ratio, then, is

$$\begin{aligned}&\frac{1+\ln(1/\beta)}{(1-\beta)(1-\alpha)}\max\left\{r \ln \frac{1}{\beta}, \ln \frac{n}{\alpha}\right\} \\ &\leq \frac{2}{1-1/e}\left(1+\frac{1}{r}\right)\max\{r, \ln(n(r+1))\} \\ &= \frac{2}{1-1/e}\max\left\{(r+1), \left(1+\frac{1}{r}\right)\ln(n(r+1))\right\} .\end{aligned}$$

We can continue, using the facts that $r \leq \gamma \ln(n(r+1)))$ and $r \geq 1$.

$$\begin{aligned}&\frac{2}{1-1/e}\max\left\{(r+1), \left(1+\frac{1}{r}\right)\ln(n(r+1))\right\} \\ &\leq \frac{2}{1-1/e}\max\{\gamma \ln(n(r+1))+1, 2\ln(n(r+1))\} \\ &= \frac{2}{1-1/e}(\gamma \ln(n(r+1))+1) \\ &\leq \frac{2}{1-1/e}(\gamma \ln(n(r+1))+1) \\ &\leq 3.2\gamma \ln(n(r+1))+3.2\end{aligned}$$

The additive part is identical except that the $\max\{\cdots\}$ is replaced by $\ln \frac{n}{\alpha}$, a factor of γ less. ■

Thus, using Share, we can achieve our $poly(L, \log n)$ ratio for L-depth HSTs. But we reach our $O(\log^5 n \log \log n)$ bound using a different unfair MTS algorithm called Odd-Exponent. We turn to examining Odd-Exponent and using it to build a MTS algorithm with a $polylog(n)$ competitive ratio.

Chapter 4

A general-metric MTS algorithm

This chapter presents the $polylog(n)$-competitive algorithm for metrical task systems. We begin by examining a different algorithm Odd-Exponent for the r-unfair uniform MTS problem. Interestingly, although Odd-Exponent and Share are radically different in approach, they share similar guarantees. Share is the simpler and more intuitive algorithm, but Odd-Exponent is an interesting alternative with slightly more efficient MTS guarantees. In particular, with Odd-Exponent we can guarantee a $O(\log^5 n \log\log n)$ competitive ratio on general metric spaces, whereas using Share gives us instead $O(\log^7 n \log\log n)$. (The difference is that Odd-Exponent has a smaller additive part in its guarantee.)

4.1 Linear

For intuition, we first consider what we should do for two regions. One very good strategy (in fact, the optimal r-unfair strategy) is to allocate to region 1 the probability

$$p_1 = \frac{1}{2} + \frac{\mathbf{OPT}_2 - \mathbf{OPT}_1}{2} .$$

and to region 2 the remainder. This is the strategy that Blum *et al.* use for equal-ratio regions [BKRS92]. Its r-unfair competitive ratio is $r + 1$; the derivation, analysis, and proof of optimality is identical to the approach we later see in Theorem 4.7.

For more than 2 regions, the natural approach is to generalize the 2-region equation. We call this algorithm Linear to emphasize the linear movement of probability as the work function changes.

Algorithm Linear We allocate to region j the probability

$$p_j = \frac{1}{n} + \frac{1}{n}\sum_{i \neq j}(\mathbf{OPT}_i - \mathbf{OPT}_j) .$$

The following analysis of Linear is simpler than the later Odd-Exponent analysis, but it follows the same basic method.

To simplify our analysis of these algorithms, we employ two assumptions. The first is to assume that each task vector is 0 in all components, except one component which is bounded by δ. We can choose δ to be as small as we want. Such a task is called an **elementary task** or a **δ-elementary task**. The following theorem, not proven here, justifies this assumption.

Theorem 4.1 ([Tom97, BEY98]) *For any metric space and any $\delta > 0$, if we have a ρ-competitive* MTS *algorithm assuming δ-elementary task vectors, then we can construct a ρ-competitive* MTS *algorithm.*

We use the notation (j, δ) to represent a task where j is the state incurring a cost of δ.

Our second assumption is the following.

Assumption 4.1 *For an elementary task giving a cost of δ to a state v so that all probability on v is removed, we can assume that δ is the least value causing the algorithm to do this.*

This is because a larger δ does not alter the on-line cost, although it may increase the off-line cost. The end of Section 6.3 (which presents results of an empirical comparison of several unfair MTS algorithms including Odd-Exponent and Share) discusses how an implementation can efficiently incorporate these assumptions.

Theorem 4.2 *The r-unfair competitive ratio of* Linear *is at most $r + (n-1)$.*

Proof. We use a potential function

$$\Phi = \frac{r}{2n} \sum_{i,j:i \neq j} (\mathbf{OPT}_i - \mathbf{OPT}_j)^2 ,$$

and our analysis competes against the average work-function value, $\frac{1}{n}\sum_i \mathbf{OPT}_i$, which is at most 1 from the true optimum, $\min_i \mathbf{OPT}_i$.

Say we receive an elementary task vector where only a state k incurs a cost δ. Let p_k and p_k' represent the probability in region k before and after the task vector, and let Φ and Φ' represent the potential before and after. Then the on-line strategy's amortized cost is

$$p_k' r\delta + (p_k - p_k') + \Phi' - \Phi .$$

Assumption 4.1 implies that $\mathbf{OPT}_k$ will rise by exactly δ. Because p_k decreases as a function of $\mathbf{OPT}_k$, we can upper-bound this cost using an integral.

$$\int_y^{y+\delta} \left(p_k r - \frac{\partial p_k}{\partial \mathbf{OPT}_k} + \frac{\partial \Phi}{\partial \mathbf{OPT}_k} \right) d\mathbf{OPT}_k$$

We compute the integrand.

$$\begin{aligned}
& p_k r - \frac{\partial p_k}{\partial \mathbf{OPT}_k} + \frac{\partial \Phi}{\partial \mathbf{OPT}_k} \\
= & \left(\frac{r}{n} + \frac{r}{n} \sum_{i \neq k} (\mathbf{OPT}_i - \mathbf{OPT}_k) \right) - \left(\frac{1}{n} \sum_{i \neq k} (-1) \right) + \frac{r}{n} \sum_{i \neq k} (\mathbf{OPT}_k - \mathbf{OPT}_i) \\
= & \frac{r + n - 1}{n}
\end{aligned}$$

Thus the total cost is

$$\int_y^{y+\delta} \frac{r+n-1}{n} \, d\mathbf{OPT}_k = \frac{\delta}{n}(r + n - 1) ,$$

which is $r + (n-1)$ times the change in $\frac{1}{n}\sum_i \mathbf{OPT}_i$ of $\frac{\delta}{n}$. ■

4.2 Odd-Exponent

Although $r + (n-1)$ is an interesting alternative to the $(r+1)H_n$ guarantee of Marking, it falls short of what we need. By adding a parameter t to Linear in a peculiar way, it turns out that we get the best of both worlds.

Before discussing the strategy, we first define the **odd exponent function**, notated $x^{[t]}$ for any $x \in \mathbb{R}$ and $t \geq 0$.

$$x^{[t]} = \begin{cases} x^t & \text{if } x \geq 0 \\ -(-x)^t & \text{if } x < 0 \end{cases}$$

In our analysis, we use the relationship in the derivatives of $x^{[t]}$ and $|x|^t$ (which we could term the *even exponent function*) for $t > 1$.

$$\begin{aligned} \frac{d}{dx}|x|^t &= tx^{[t-1]} \\ \frac{d}{dx}x^{[t]} &= t\,|x|^{t-1} \end{aligned}$$

Note also that

$$\begin{aligned} x^{[t]} + |x|^t &= \begin{cases} 2x^t & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \qquad (4.1) \\ x^{[t]} + (-x)^{[t]} &= 0 \end{aligned}$$

Algorithm Odd-Exponent The strategy uses a parameter $t \geq 1$. (Think $t = O(\log n)$.) We allocate to region j the probability

$$p_j = \frac{1}{n} + \frac{1}{n}\sum_{i=1}^{n} (\mathbf{OPT}_i - \mathbf{OPT}_j)^{[t]} \ . \qquad (4.2)$$

Lemma 4.3 *Odd-Exponent maintains legal probability distributions ($\sum_j p_j = 1$ and each p_j is nonnegative).*

Proof. It maintains $\sum_j p_j = 1$, since because $x^{[t]}$ is an odd function, $\sum_j \sum_i (\mathbf{OPT}_i - \mathbf{OPT}_j)^{[t]} = 0$. Because p_j is a decreasing function of only $\mathbf{OPT}_j$ among the $\mathbf{OPT}$ values, Assumption 4.1 implies that each p_j remains nonnegative. (Requests to $i \neq j$ only increase p_j. Say we receive a request (j, δ) that would make p_j negative if $\mathbf{OPT}_j$ increased by δ. Since the distribution (4.2) is continuous, there is an $\delta' < \delta$ for which the algorithm sets p_j to zero. Assumption 4.1 implies that we can use (j, δ') instead so that p_j becomes exactly zero.) ■

In the remainder of this section we analyze the strategy's r-unfair competitive ratio and then its additive part.

To analyze the performance we require a simple general lemma.

Lemma 4.4 *Consider n nonnegative reals $x_1, \ldots, x_n$ and two numbers $1 \leq s \leq t$. If $\sum_i x_i^t \leq 1$, then $\sum_i x_i^s \leq n^{(t-s)/t}$.*

This lemma, presented here without proof, is not difficult to understand. The value of $\sum_i x_i^s$ is maximum when all the terms are equal.

Theorem 4.5 *The r-unfair competitive ratio of Odd-Exponent is at most $r + 2n^{1/t}t$.*

Remark. If we choose t to be $\ln n$, this ratio translates to $r + 2e \ln n$.

Proof. We use two potential functions Φ_ℓ and Φ_m. The potential function Φ_ℓ amortizes the *local* cost within each region.

$$\Phi_\ell = \frac{r}{2(t+1)n} \sum_i \sum_j |\mathbf{OPT}_i - \mathbf{OPT}_j|^{t+1}$$

The other potential, Φ_m, amortizes the *movement* cost between regions.

$$\Phi_m = \frac{1}{2n} \sum_i \sum_j |\mathbf{OPT}_i - \mathbf{OPT}_j|^{t}$$

The potential Φ for the strategy is simply $\Phi_\ell + \Phi_m$.

Justified by Theorem 4.1 and Assumption 4.1, we assume that, for a request (k, δ), $\mathbf{OPT}_k$ increases from some value y to $y + \delta$. In this analysis the strategy competes against the average $\mathbf{OPT}$ value, $\frac{1}{n}\sum_i \mathbf{OPT}_i$. So the off-line cost is $\frac{\delta}{n}$.

Let p_k and p'_k represent the probability in region k before and after the task vector, and let Φ_ℓ (Φ_m) and Φ'_ℓ (Φ'_m) represent the local (movement) potential before and after the task vector. Then the on-line strategy's cost is

$$p'_k r\delta + (p_k - p'_k) + \Phi'_\ell + \Phi'_m - \Phi_\ell - \Phi_m \ .$$

Because p_k decreases as a function of $\mathbf{OPT}_k$, we can upper-bound this cost using an integral.

$$\int_y^{y+\delta} \left(p_k r + \frac{\partial \Phi_\ell}{\partial \mathbf{OPT}_k} - \frac{\partial p_k}{\partial \mathbf{OPT}_k} + \frac{\partial \Phi_m}{\partial \mathbf{OPT}_k} \right) d\mathbf{OPT}_k \tag{4.3}$$

We examine the first two terms, representing the local cost, and the last two terms, representing the movement cost, separately. In particular, we show that the amortized local cost is at most r/n, while the amortized movement cost is at most $2n^{1/t}t/n$.

For the local cost, notice that, for any j,

$$\frac{\partial \Phi_\ell}{\partial \mathbf{OPT}_j} = -\frac{r}{n} \sum_i (\mathbf{OPT}_i - \mathbf{OPT}_j)^{[t]} = -\left(p_j - \frac{1}{n}\right) r \ .$$

Thus the local cost terms are equal to r/n.

$$p_k r + \frac{\partial \Phi_\ell}{\partial \mathbf{OPT}_k} = p_k r - \left(p_k - \frac{1}{n}\right) r = \frac{r}{n} \ . \tag{4.4}$$

Analyzing the movement cost requires more work.

$$\begin{aligned} -\frac{\partial p_k}{\partial \mathbf{OPT}_k} + \frac{\partial \Phi_m}{\partial \mathbf{OPT}_k} &= \frac{t}{n} \sum_{i \neq k} |\mathbf{OPT}_i - \mathbf{OPT}_k|^{t-1} + \frac{t}{n} \sum_{i \neq k} (\mathbf{OPT}_k - \mathbf{OPT}_i)^{[t-1]} \\ &= \frac{2t}{n} \sum_{\mathbf{OPT}_i < \mathbf{OPT}_k} (\mathbf{OPT}_k - \mathbf{OPT}_i)^{t-1} \end{aligned} \tag{4.5}$$

The last step follows from equation (4.1). We would like to simplify the summation. Say that $\mathbf{OPT}_a$ is currently the maximum $\mathbf{OPT}$ value. Observe using the probability allocation (4.2) that, since p_a is not negative, the following holds.

$$\sum_{i \neq a} (\mathbf{OPT}_a - \mathbf{OPT}_i)^t = \sum_{i \neq a} (\mathbf{OPT}_a - \mathbf{OPT}_i)^{[t]} \leq 1 \tag{4.6}$$

Because $\mathbf{OPT}_a$ is maximum, each term of the summation is positive. Thus it follows from Lemma 4.4 that

$$\sum_{i \neq a} (\mathbf{OPT}_a - \mathbf{OPT}_i)^{t-1} \leq (n-1)^{1/t} < n^{1/t}\,.$$

Using the definition of a again we continue from equation (4.5) to finish approximating the movement cost.

$$\frac{2t}{n} \sum_{\mathbf{OPT}_i < \mathbf{OPT}_k} (\mathbf{OPT}_k - \mathbf{OPT}_i)^{t-1} \leq \frac{2t}{n} \sum_{i \neq a} (\mathbf{OPT}_a - \mathbf{OPT}_i)^{t-1} < \frac{2n^{1/t}t}{n} \tag{4.7}$$

The estimates of the local cost (4.4) and movement cost (4.7) bound the total cost (4.3) by

$$\begin{aligned}
\int_y^{y+\delta} \left(p_k r + \frac{\partial \Phi_\ell}{\partial \mathbf{OPT}_k} - \frac{\partial p_k}{\partial \mathbf{OPT}_k} + \frac{\partial \Phi_m}{\partial \mathbf{OPT}_k} \right) d\mathbf{OPT}_k& \\
\leq \int_y^{y+\delta} \frac{r + 2n^{1/t}t}{n}\, d\mathbf{OPT}_k& \\
= \frac{\delta}{n}(r + 2n^{1/t}t)&
\end{aligned}$$

The off-line cost (according to $\frac{1}{n} \sum_j \mathbf{OPT}_j$) is $\frac{\delta}{n}$, so the amortized competitive ratio is $r + 2n^{1/t}t$ as desired. ∎

To apply Odd-Exponent recursively on a k-HST, we must also bound the additive part for its r-unfair ratio. We see when we do this that we may want to choose a large value for t since it reduces the maximum potential.

Lemma 4.6 *The additive part to the ratio in Theorem 4.5 is bounded by* $\frac{r}{t+1} + 2$.

Proof. The additive part is the maximum change in potential from the beginning, plus 1 because the proof of Theorem 4.5 competes relative to $\frac{1}{n} \sum_i \mathbf{OPT}_i$, which may be as much as 1 away from $\min_i \mathbf{OPT}_i$. First, we bound Φ_ℓ. Let a be the index of the maximum $\mathbf{OPT}$ value.

$$\begin{aligned}
\Phi_\ell &= \frac{r}{2(t+1)n} \sum_i \sum_j |\mathbf{OPT}_i - \mathbf{OPT}_j|^{t+1} \\
&= \frac{r}{(t+1)n} \sum_i \sum_{\mathbf{OPT}_j < \mathbf{OPT}_i} (\mathbf{OPT}_i - \mathbf{OPT}_j)^{t+1} \\
&\leq \frac{r}{(t+1)n} \sum_i \sum_{\mathbf{OPT}_j < \mathbf{OPT}_i} (\mathbf{OPT}_a - \mathbf{OPT}_j)^{t+1} \\
&\leq \frac{r}{t+1} \sum_j (\mathbf{OPT}_a - \mathbf{OPT}_j)^{t+1} \\
&\leq \frac{r}{t+1} \sum_j (\mathbf{OPT}_a - \mathbf{OPT}_j)^{t} \qquad (4.8) \\
&\leq \frac{r}{t+1} \qquad (4.9)
\end{aligned}$$

Inequality (4.8) follows because, since $\mathbf{OPT}_a \leq \mathbf{OPT}_j + 1$, each term of the summation is at most one, so reducing the term's exponent increases the term's value. Inequality (4.9) comes from equation (4.6).

Bounding Φ_m is similar. Again, let a be the index of the maximum $\mathbf{OPT}$ value.

$$\begin{aligned}\Phi_m &= \frac{1}{2n}\sum_i\sum_j |\mathbf{OPT}_i - \mathbf{OPT}_j|^t \\ &= \frac{1}{n}\sum_i \sum_{\mathbf{OPT}_j<\mathbf{OPT}_i} (\mathbf{OPT}_i - \mathbf{OPT}_j)^t \\ &\le \frac{1}{n}\sum_i \sum_{\mathbf{OPT}_j<\mathbf{OPT}_i} (\mathbf{OPT}_a - \mathbf{OPT}_j)^t \\ &\le \sum_j (\mathbf{OPT}_a - \mathbf{OPT}_j)^t \\ &\le 1\end{aligned}$$

Adding this to the bound for Φ_ℓ in equation (4.9) gives the total bound on the potential. To bound the additive part, we add 1 more because $\frac{1}{n}\sum_i \mathbf{OPT}_i$ may differ from $\min_i \mathbf{OPT}_i$ by as much as 1. ■

4.3 Two-Region

Currently we have a technique (actually, two) for guaranteeing a $poly(L, \log n)$ ratio for HSTs, where L is the depth of the tree. It would be nice if we could guarantee that $L = polylog(n)$, and indeed for many restricted sets of metric spaces we can; but such a guarantee for general metric spaces is impossible to make. For example, if we lay points at $1, 2, 4, \ldots, 2^{n-1}$ on an axis, the resulting h-HST must have depth $\Omega(\log_h 2^n)$. Thus, although we have made solid progress toward the $polylog(n)$ ratio, we need new ideas to achieve it. These appear in the remainder of this chapter.

The main remaining idea is more of a convoluted hack than an elegant, final answer. The idea is simple: A tree with more than $polylog(n)$ levels must have nodes whose subtrees are very unbalanced — one subtree has many more leaves than any of the others. Or, since competitive ratios are strongly tied to the tree size, we can reword it in the jargon of unfairness: We want to handle the case where the cost ratios are different for different points in the space. (We have until now always assumed they are equal for all points.)

Having different cost ratios for different points appears to be a complex issue. But there is one particularly simple case that we can tackle: the case of having only two points with separate cost ratios. We can utilize this in building a strategy for the HST: Where the subtrees are all roughly the same size, we can still use Odd-Exponent profitably; but where one is much larger, we can combine all but the largest using Odd-Exponent and then apply Two-Region to combine this combination with the largest subtree.

We first look at the unusual two-point unfair scenario and present Two-Region as our algorithm. In this problem, one point has unfairness r_1 while the other has unfairness r_2. Blum *et al.* consider this scenario, but their analysis does not have to worry about the additive constant [BKRS92]. Seiden [Sei99] independently develops the same algorithm.

Algorithm Two-Region ([BKRS92, Sei99]) Without loss of generality, say $r_1 > r_2$. Let β represent $e^{r_1 - r_2}$, and define $p_1(y)$ as follows.

$$p_1(y) = \frac{\beta - \beta^{\frac{1}{2}+\frac{y}{2}}}{\beta - 1} \tag{4.10}$$

After computing the work function $\mathbf{OPT}$, we place $p_1(\mathbf{OPT}_1 - \mathbf{OPT}_2)$ probability in the first region and the rest in the second.

While the strategy is hardly intuitive, the analysis will make the reason for the selection clear.

Theorem 4.7 *The competitive ratio of* Two-Region *is*

$$r_1 + \frac{r_1 - r_2}{e^{r_1 - r_2} - 1}$$

The additive part is at most $r_2 + 2$.

Proof. Because $p_1(1) = 0$ and $p_1(-1) = 1$, this algorithm does not have the problem of allocating nonnegative probability to a pinned state.

What we will show is that for a given potential $\mathbf{\Phi}$, for any task t, the cost to Two-Region is bounded by

$$cost_{\mathsf{Two-Region}} + \left(\mathbf{\Phi}^t - \mathbf{\Phi}^{t-1}\right) \leq \left(r_1 + \frac{r_1 - r_2}{\beta - 1}\right)\left(\mathbf{OPT}_1^t - \mathbf{OPT}_1^{t-1}\right) .$$

This means that to achieve the ratio, the potential must entirely absorb the cost any time the second state incurs some cost. We define the potential, therefore, as

$$\Phi(y) = (1 - p_1(y)) + r_2 \int_{-1}^{y} (1 - p_1(y))\, dy\,,$$

and the potential $\mathbf{\Phi}^t$ as $\Phi(\mathbf{OPT}_1^t - \mathbf{OPT}_2^t)$. This potential completely absorbs all increases to $\mathbf{OPT}_2$.

Let us consider a request that increases $\mathbf{OPT}_1$ from z to $z + \delta$. The strategy's amortized cost for this request is at most

$$\int_z^{z+\delta} \left(p_1(y)r_1 - \frac{dp_1}{dy} + \frac{d\Phi}{dy}\right) dy \quad \leq \quad \int_z^{z+\delta} \left(p_1(y)r_1 - 2\frac{dp_1}{dy} + (1 - p_1(y))r_2\right)\, dy$$

(The integral approximates the cost because p_1 is a decreasing function.) By setting this to a constant we obtain a first-order differential equation in p_1, which can be solved with the boundary conditions $p_1(1) = 0$ and $p_1(-1) = 1$. The solution is as in equation (4.10). It is easy to verify that this results in a constant integrand.

$$\begin{aligned}\int_z^{z+\delta} \left(p_1(y)r_1 - 2\frac{dp_1}{dy} + (1 - p_1(y))r_2\right)\, dy \quad &= \quad \int_z^{z+\delta} \left(r_1 + \frac{r_1 - r_2}{\beta - 1}\right)\, dy \\ &= \quad \left(r_1 + \frac{r_1 - r_2}{\beta - 1}\right)\delta\end{aligned}$$

Since the off-line player pays δ, the competitive ratio for the strategy is as advertised.

To bound the additive part, we note how widely the potential can vary. Because always $y \geq -1$, the potential is always nonnegative. The potential is largest when $y = 1$. In this case the first term is 1 and (using some straightforward calculus) the second term is at most r_2. Thus the potential is at most $r_2 + 1$. Since $\mathbf{OPT}_1$ differs from the optimal cost by at most 1, the additive part is at most $r_2 + 2$. ∎

4.4 Building the $polylog(n)$ algorithm

As in Theorem 2.7, we build our algorithm for the HST inductively. In building the algorithm, we modify the HST so that the distance between any two points does not decrease but may increase to twice the initial distance. This costs us only a factor of 2 in the overall ratio.

Theorem 4.8 *For an h-HST with $h \geq 8000 \ln^2 n$, we can modify the HST so that distances at most double and for the modification we have an on-line MTS algorithm with a competitive ratio of at most $1000 \ln^2 n$ with an additive $2000D \ln^2 n$, where D is the diameter of the modified tree.*

Remark. The following proof draws heavily on the technical details already discussed in Theorem 2.7. Understanding that proof is essential to understanding the following.

To avoid complications, this theorem employs intentionally generous constants.

Proof. We prove this inductively on the tree, with the base case being the trivial single-node tree. For the induction step, we let r_i be the ratio for subtree i, with the subtrees ordered so $r_1 \geq \cdots \geq r_b$. We define n_i as the number of points in subtree i. Finally, n is the total number of points in all subtrees.

The induction step has two cases, depending on whether r_1 is below $1000 \ln^2 n - 50 \ln n$ (in which case the subtrees are balanced enough to simply apply Odd-Exponent) or above (in which case the subtrees are very unbalanced) .

Case 1. If $r_1 < 1000 \ln^2 n - 50 \ln n$, then our strategy is to first mutate the tree by doubling the distances from the root node to the points. We apply Odd-Exponent to combine the subtrees using $t = \ln n$.

To bound the competitive ratio, we observe that for an arbitrary action sequence $\mathbf{v}$, implying an action sequence $\mathbf{v}_b$ for moving between subtrees, the off-line cost is at least

$$\mathit{local}(\mathbf{T}_A, \mathbf{v}_b) + \left(D - \frac{D}{h}\right) \mathit{move}(\mathbf{v}_b) = \mathit{local}(\mathbf{T}_A, \mathbf{v}_b) + \zeta D \, \mathit{move}(\mathbf{v}_b) ,$$

where we define ζ as $1 - \frac{1}{h}$. Meanwhile, given the action sequence $\mathbf{v}_A$ used by the on-line algorithm, the expected cost is at most

$$\begin{aligned} &\left(1000 \ln^2 n - 50 \ln n\right) \mathit{local}(\mathbf{T}_A, \mathbf{v}_A) + \left(D + 2000 \frac{D}{h} \ln^2 n\right) \mathit{move}(\mathbf{v}_A) + 2000 \frac{D}{h} \ln^2 n \\ &\quad \leq \quad \left(1000 \ln^2 n - 50 \ln n\right) \mathit{local}(\mathbf{T}_A, \mathbf{v}_A) + \frac{5D}{4} \mathit{move}(\mathbf{v}_A) + \frac{D}{4} . \end{aligned}$$

We can use Odd-Exponent's unfair competitive ratio to bound the on-line cost in expectation over Odd-Exponent's random choices.

$$\begin{aligned} &\mathbf{E}\left[\left(1000 \ln^2 n - 50 \ln n\right) \mathit{local}(\mathbf{T}_A, \mathbf{v}_A) + \frac{5D}{4} \mathit{move}(\mathbf{v}_A) + \frac{D}{4}\right] \\ &= \quad \frac{5D}{4} \mathbf{E}\left[\left(800\zeta \ln^2 n - 40\zeta \ln n\right) \mathit{local}\left(\frac{1}{\zeta D} \mathbf{T}_A, \mathbf{v}_A\right) + \mathit{move}(\mathbf{v}_A)\right] + \frac{D}{4} \\ &\leq \quad \frac{5D}{4} \left(\left(800\zeta \ln^2 n - 40\zeta \ln n + 5.5 \ln n\right) \left(\mathit{local}\left(\frac{1}{\zeta D} \mathbf{T}_A, \mathbf{v}_b\right) + \mathit{move}(\mathbf{v}_b)\right)\right. \\ &\qquad \left. + \frac{800\zeta \ln^2 n - 40\zeta \ln n}{\ln n + 1} + 2\right) + \frac{D}{4} \\ &\leq \quad \left(1000 \ln^2 n\right) \left(\mathit{local}(\mathbf{T}_A, \mathbf{v}_b) + \zeta D \, \mathit{move}(\mathbf{v}_b)\right) + 1000\zeta D \ln n + \frac{D}{4} \end{aligned}$$

Thus we have satisfied our inductive hypothesis.

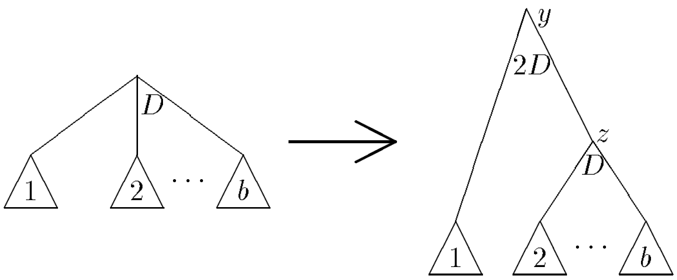


Figure 4.1: Transformation for Case 2 of Theorem 4.8.

Case 2. If $r_1 \geq 1000 \ln^2 n - 50 \ln n$, then our strategy is to first mutate the tree by splitting the root node into two nodes y and z, where subtrees 2 through b are subtrees of z, while the subtrees of y are subtree 1 and the tree rooted at z. (Figure 4.1 illustrates this.) The distances in the tree rooted at z remain the same, but y is lifted so that its distance from the leaves doubles. At z, the ratio of the largest subtree's diameter to the overall tree's diameter is at most $\frac{h}{2}$ (since we doubled the diameter of the subtrees in the inductive step); at y, this ratio is 2.

Our algorithm is to use Odd-Exponent to combine the subtrees of z (choosing $t = 2 \ln n$), and to use Two-Region to combine the subtrees of y. To analyze the competitive ratio, we first analyze the tree rooted at z and then the tree rooted at y. We assign x so that $n_1 = (1 - \frac{1}{x})n$; since $r_1 \geq 1000 \ln^2 n - 50 \ln n$ and $r_1 \leq 1000 \ln^2 n_1$, we can deduce that $40 \leq x \leq n$.

For the tree rooted at z, we observe that for an arbitrary action sequence $\mathbf{v}$, implying an action sequence $\mathbf{v}_b$ for moving between subtrees, the off-line cost is at least

$$local(\mathbf{T}_A, \mathbf{v}_b) + \left(D - \frac{2D}{h}\right) move(\mathbf{v}_b) = local(\mathbf{T}_A, \mathbf{v}_b) + \zeta D\, move(\mathbf{v}_b)\,,$$

where we define ζ as $1 - \frac{2}{h}$. Meanwhile, given the action sequence $\mathbf{v}_A$ used by the on-line algorithm, the expected cost is at most

$$\begin{aligned}
&\left(1000 \ln^2 \frac{n}{x}\right) local(\mathbf{T}_A, \mathbf{v}_A) + \left(D + 2000\frac{2D}{h} \ln^2 \frac{n}{x}\right) move(\mathbf{v}_A) + 2000\frac{2D}{h} \ln^2 \frac{n}{x} \\
&\qquad \leq \left(1000 \ln^2 \frac{n}{x}\right) local(\mathbf{T}_A, \mathbf{v}_A) + \frac{3D}{2} move(\mathbf{v}_A) + \frac{D}{2}\,.
\end{aligned}$$

We can use Odd-Exponent's unfair competitive ratio to bound the on-line cost in expectation over Odd-Exponent's random choices.

$$\begin{aligned}
&\mathbf{E}\left[\left(1000 \ln^2 \frac{n}{x}\right) local(\mathbf{T}_A, \mathbf{v}_A) + \frac{3D}{2} move(\mathbf{v}_A) + \frac{D}{2}\right] \\
&= \frac{3D}{2}\mathbf{E}\left[\left(\frac{2000\zeta}{3} \ln^2 \frac{n}{x}\right) local\left(\frac{1}{\zeta D}\mathbf{T}_A, \mathbf{v}_A\right) + move(\mathbf{v}_A)\right] + \frac{D}{2} \\
&\leq \frac{3D}{2}\left(\left(\frac{2000\zeta}{3} \ln^2 \frac{n}{x} + 6.6 \ln n\right)\left(local\left(\frac{1}{\zeta D}\mathbf{T}_A, \mathbf{v}_b\right) + move(\mathbf{v}_b)\right) + \frac{2000\zeta \ln^2 \frac{n}{x}}{6 \ln n} + 2\right) + \frac{D}{2} \\
&\leq \left(1000 \ln^2 \frac{n}{x} + \frac{10}{\zeta} \ln n\right)(local(\mathbf{T}_A, \mathbf{v}_b) + \zeta D move(\mathbf{v}_b)) + 500\zeta D \ln n + \frac{7}{2}D
\end{aligned}$$

Thus the competitive ratio r_z for the tree rooted at z is at most $1000 \ln^2 \frac{n}{x} + \frac{10}{\zeta} \ln n$, with an additive part of at most $500\zeta D \ln n + \frac{7}{2}D$.

For an arbitrary action sequence $\mathbf{v}$, implying an action sequence $\mathbf{v}_b$ moving between the two subtrees of y, the off-line cost is at least

$$local(\mathbf{T}_A, \mathbf{v}_b) + \frac{D}{2} move(\mathbf{v}_b) \ .$$

For the on-line algorithm, given that it uses the action sequence $\mathbf{v}_A$, the expected cost is at most

$$\begin{aligned} & r_1 local_1(\mathbf{T}_A, \mathbf{v}_A) + r_z local_2(\mathbf{T}_A, \mathbf{v}_A) + \left(D + \frac{1}{2}\left(500\zeta D \ln n + \frac{7}{2}D + 500\frac{D}{h}\ln^2 n\right)\right) move(\mathbf{v}_A) \\ & \quad + 500\zeta D \ln n + \frac{7}{2}D + 500\frac{D}{h}\ln^2 n \\ \leq \ & r_1 local_1(\mathbf{T}_A, \mathbf{v}_A) + r_z local_2(\mathbf{T}_A, \mathbf{v}_A) + \left(250 D \ln n + \frac{11}{4}D\right) move(\mathbf{v}_A) + 500 D \ln n + \frac{7}{2}D \ . \end{aligned}$$

Here $local_1(\mathbf{T}_A, \mathbf{v}_A)$ represents the total cost incurred at point 1 with the task sequence $\mathbf{T}_A$ using the action sequence $\mathbf{v}_A$. (The peculiar movement cost comes from the fact that half of the movements involve the additive cost of $500\frac{D}{h}\ln^2 n$ and half involve the additive cost $500\zeta D \ln n + \frac{11}{4}D$.) We can find the expectation over Two-Region's selection of $\mathbf{v}_A$ by using the competitive ratio of Two-Region. In the following, we let α represent $250 \ln n + \frac{11}{4}$.

$$\begin{aligned} & \mathbf{E}\left[r_1 local_1(\mathbf{T}_A, \mathbf{v}_A) + r_z local_2(\mathbf{T}_A, \mathbf{v}_A) + \alpha D move(\mathbf{v}_A) + 500 D \ln n + \frac{7}{2}D\right] \\ = \ & \alpha D \mathbf{E}\left[\frac{r_1}{2\alpha} local_1\left(\frac{2}{D}\mathbf{T}_A, \mathbf{v}_A\right) + \frac{r_z}{2\alpha} local_2\left(\frac{2}{D}\mathbf{T}_A, \mathbf{v}_A\right) + move(\mathbf{v}_A)\right] + 500 D \ln n + \frac{7}{2}D \\ \leq \ & \alpha D \left(\left(\frac{r_1}{2\alpha} + \frac{\frac{r_1}{2\alpha} - \frac{r_z}{2\alpha}}{e^{\frac{r_1}{2\alpha} - \frac{r_z}{2\alpha}} - 1}\right)\left(local_1\left(\frac{2}{D}\mathbf{T}_A, \mathbf{v}_A\right) + local_2\left(\frac{2}{D}\mathbf{T}_A, \mathbf{v}_A\right) + move(\mathbf{v}_A)\right) + \left(2\frac{r_z}{2\alpha} + 1\right)\right) \\ & \quad + 500 D \ln n + \frac{7}{2}D \\ = \ & \left(r_1 + \frac{r_1 - r_z}{e^{\frac{r_1 - r_z}{2\alpha}} - 1}\right)\left(local_1(\mathbf{T}_A, \mathbf{v}_A) + local_2(\mathbf{T}_A, \mathbf{v}_A) + \frac{D}{2} move(\mathbf{v}_A)\right) \\ & \quad + (r_z D + \alpha D) + 500 D \ln n + 2D \end{aligned}$$

Thus our computed competitive ratio is $r_1 + (r_1 - r_z)/(e^{(r_1 - r_z)/2\alpha} - 1)$. We want to bound this by $1000 \ln^2 n$. To do this, we first bound $r_1 - r_z$ from below (since $x/(e^{x/2\alpha} - 1)$ decreases as x increases beyond 2α).

$$\begin{aligned} r_1 - r_z \ & \geq \ (1000 \ln^2 n - 50 \ln n) - \left(1000 \ln^2 \frac{n}{x} + \frac{10}{\zeta} \ln n\right) \\ & = \ 2000 \ln x \ln n - 1000 \ln^2 x - 50 \ln n - \frac{10}{\zeta} \ln n \\ & \geq \ 900 \ln x \ln n \end{aligned}$$

We use this to bound the ratio.

$$\begin{aligned} r_1 + \frac{r_1 - r_z}{e^{\frac{r_1 - r_z}{500 \ln n + 4}} - 1} \ & \leq \ 1000 \ln^2 \left(\left(1 - \frac{1}{x}\right) n\right) + \frac{900 \ln x \ln n}{e^{\frac{900 \ln x \ln n}{500 \ln n + 4}} - 1} \\ & \leq \ 1000 \ln^2 n - \frac{2000}{x} \ln n + \frac{1000}{x^2} + \frac{900 \ln x \ln n}{e^{1.8 \ln x} - 1} \\ & \leq \ 1000 \ln^2 n \end{aligned}$$

Likewise, we can bound the additive part

$$\left(r_z + 250\ln n + \frac{11}{4} + 500\ln n + \frac{7}{2}\right) D \ ,$$

which is less than $2000D\ln^2 n$.

■

4.5 Extensions

We can extend Theorem 4.8 in two ways: We can try using Share instead of Odd-Exponent, and we can look at what happens in specific metric spaces.

An alternative algorithm

Theorem 4.8 did not rely on any specific properties of Odd-Exponent. But if we were to apply Share instead, the ratio would suffer due to the additive part: Share has an additive r, while Odd-Exponent has an additive $r/\log n$. The $r/\log n$ additive part is necessary in Case 2 to get a manageable additive part for the tree rooted at z.

When we adapted Share to MTS (Theorem 3.5), we had a parameter γ, and in fact the additive part was $\frac{r}{\gamma}$. Taking $\gamma = \log n$, the unfair competitive ratio is $r + \log^2 n$. So if we use Share in proving an $O(\log^3 n)$ bound on an $\Omega(\log^3 n)$-HST, we can get a working theorem. The net result is an $O(\log^7 n \log\log n)$ bound for general metric spaces.

Alternative spaces

For many restricted sets of metric spaces, the bound improves by using Odd-Exponent (or Share, with a penalty due to the additive part) on Theorem 2.7. For example, if the metric between states comes from the shortest-path metric on an unweighted graph on states, we know the depth of the HST must be $O(\log_h n)$, so we can get a ratio of $O(\log^2 n/\log\log n)$ on an $\Omega(\log n)$-HST. Since we can probabilistically $O(\log^2 n \log\log n)$-approximate unweighted graphs by $\Omega(\log n)$-HSTs, we get a result of $O(\log^4 n)$ for unweighted graphs.

We can do even better for HSTs that are "roughly balanced" in the sense that, at any node with b subtrees covering a total of n nodes, the largest subtree contains $1.4\frac{n}{b}$ nodes. In this case, we can use the inductive hypothesis that the ratio is $4e\ln n$: The largest subtree has a ratio r of at most $4e\ln 1.4\frac{n}{b}$, and so Odd-Exponent combining the b subtrees has ratio $r + 2e\ln b \le 4e\ln n$. The additive part is $O(\log n)$, so we require an $\Omega(\log n)$-HST for this to work.

This "roughly balanced" property arises in mesh spaces, like a line space: We can $O(\log^2 n/\log\log n)$-approximate such a space with roughly balanced $\Omega(\log n)$-HSTs [Bar96]. Thus for mesh spaces, we get a ratio of $O(\log^3 n/\log\log n)$.

Chapter 5

Combining on-line algorithms

Now we switch away from the general-metric MTS problem; instead we pick up on the theme of Section 3.4 and extend the application of Experts algorithms to competitive analysis. In each of Chapters 5, 6, and 7, we extend the result in a different way; in this chapter, we examine applications to the problem of *combining on-line algorithms on-line*.

Problem Combine-Online Given are a variety of on-line algorithms $A_1, A_2, \ldots, A_n$, each incurring losses during each time step. At all times, our on-line algorithm chooses to follow one of these algorithms, incurring that algorithm's losses, but between time steps the on-line algorithm may choose to switch between algorithms at a **switching cost** of d. Our hope is that on any sequence the algorithm will not do too much worse than the best of the A_i for that sequence.

Example 5.1 Say we have a variety of paging algorithms like LRU, Marking, and MRU. On any sequence of page requests, we want to do about as well as the best among them in hindsight. One way of doing this is to follow algorithms' internal caches, allowing the on-line algorithm to switch between caches. The cost to switch between caches is at most the size of the cache (usually represented by k).

This problem is similar to one arising in Azar, Broder, and Manasse, with the difference that they do not incorporate a fixed switching cost d [ABM93]. Instead, in their problem, when the Combine-Online algorithm switches from one algorithm A_i to another A_j, the algorithm may pay as much as the total cost paid so far by A_i and by A_j. This is because they are primarily concerned with combining algorithms for the k-server problem, where the algorithms are moving within an unbounded metric space, and so in moving between algorithms the Combine-Online algorithm may have to move all the way back to the initial point of the space (which is at most the total cost paid by A_i) and then to the point currently occupied by A_j. The guarantee they achieve for this more difficult scenario is that their algorithm can guarantee it pays no more than $O(\log n)$ times the best of the on-line algorithms it is combining. (If the metric space has a bounded diameter D, then we could alternatively apply the results of this chapter to the problem using $d = D$.)

What we will see is that in our formulation, an algorithm (using Experts algorithms) can do nearly as well as the best single algorithm. In particular, if the best algorithm A_k incurs a total cost of L, then our combination algorithm will pay at most $(1 + \varepsilon)L + (1 + \frac{1}{\varepsilon})d \log n$.

Example 5.2 Another application is to the List-Update problem [BM85, ST85a]. In this problem, our algorithm maintains a list L over n elements. Each time step, the algorithms receives a request to one of the n elements and pays 1 for each step that must be made in the list to find the element. The algorithm also pays 1 each time it transposes two adjacent elements in the list, unless that transpose moves the element of the current access forward in the list.

List-Update is a classical problem in competitive analysis. Move-To-Front is one of the simplest algorithms: On each time step, Move-To-Front moves the just-accessed item to the head of the list. This algorithm has a competitive ratio of $2 - \frac{2}{n+1}$ [BM85, ST85a, Ira91]. Karp and Raghavan show that no deterministic algorithm can guarantee less, even against a static adversary (who is not allowed to alter the list ordering) (reported in [Ira91]).

The situation for randomized algorithms is less certain. The best-known algorithm is Comb, with a competitive ratio of 1.6 against dynamic adversaries [AvSW95]. No algorithm can achieve a ratio of less than $1.5 - \frac{5}{n+5}$ [Tei93]. For static adversaries, no lower bound for randomized algorithms is known, nor is there a better bound than that for Comb.

By demonstrating a (massively inefficient) algorithm, the results of this chapter imply that no such lower bound is possible for static adversaries. We can have an algorithm for each of the $n!$ possible lists; the algorithm for list L statically keeps L as its list. The switching cost between algorithms is at most $\binom{n}{2}$. From this, we get a ratio of $(1+\varepsilon)$ for any fixed ε. (This algorithm, as stated, is extraordinarily impractical. We are taking advantage of the fact that the on-line model does not count the time spent deciding which item to move. A simpler and more efficient algorithm achieving a similar guarantee would be an interesting result.)

This observation extends naturally to the Dynamic-Tree problem, where the on-line algorithm is permitted to rearrange a binary search tree by rotations along the path to the accessed node. Sleator and Tarjan demonstrate that their Splay-Tree algorithm is $O(1)$-competitive against a static adversary [ST85b]. But by having a separate algorithm for each possible tree, the results of this chapter demonstrate that one can in fact be $(1+\varepsilon)$-competitive if we do not count the time spent deciding how to rearrange the tree. (For Dynamic-Tree against dynamic adversaries, the lowest known bound is $O(\log n)$; for example, a static balanced tree achieves the bound $\lceil \lg n \rceil$.)

5.1 Simulating all algorithms

If our on-line algorithm can afford to simulate all n algorithms, then we can apply Hedge in the straightforward way: We follow the probabilities that Hedge uses. When we get an event and see how the different algorithms will process it, we give those same losses to the algorithms' corresponding experts for Hedge. We then change our probability distribution according to Hedge, moving to a new algorithm as Hedge directs.

Theorem 5.1 *Say that the best algorithm has a total loss of L. Then the loss of Hedge is at most*

$$\mathbf{E}[loss_{\mathsf{Hedge}}] \leq \frac{1 + d \ln 1/\beta}{1-\beta} \left(\left(\ln \frac{1}{\beta} \right) L + \ln n \right) .$$

Remark. Say we choose $\beta = 1 - \frac{\varepsilon}{d}$ for some $\varepsilon > 0$. Then the above bound translates to approximately $(1+\varepsilon)L + (1+\frac{1}{\varepsilon})d \ln n$. That is, we are $(1+\varepsilon)$-competitive with respect to the best on-line algorithm.

Proof. Say algorithm A_k is the best algorithm. We consider separately the local cost (that is, the amount spent by our on-line algorithm due to the algorithm it currently occupies) and the switching cost incurred by our on-line algorithm. Theorem 3.2 shows that the local cost satisfies

$$\mathbf{E}[loss_{\mathsf{Hedge}}] \leq \frac{1}{1-\beta}\left(\left(\ln\frac{1}{\beta}\right)L + \ln n\right) .$$

(In fact, we get one-step lookahead; this only decreases the algorithm's cost.)

We now show that the movement cost is at most $d\ln\frac{1}{\beta}$ times the local cost. At step t, we expect to pay $d\sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}(\mathbf{p}_i^{t-1}-\mathbf{p}_i^t)$ for movement, where $\mathbf{p}_i$ is the probability the on-line algorithm is following algorithm i at time step t.

$$\begin{aligned}
d\sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}(\mathbf{p}_i^{t-1}-\mathbf{p}_i^t) &= d\sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}-\frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^t}\right) \\
&\leq d\sum_{i:\mathbf{p}_i^{t-1}>\mathbf{p}_i^t}\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}-\frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^{t-1}}\right) \\
&\leq d\sum_i\left(\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}-\frac{\mathbf{w}_i^{t-1}\beta^{\boldsymbol{\ell}_i^t}}{\mathbf{W}^{t-1}}\right) \\
&= d\sum_i\frac{\mathbf{w}_i^{t-1}}{\mathbf{W}^{t-1}}\left(1-\beta^{\boldsymbol{\ell}_i^t}\right) \\
&\leq d\sum_i\mathbf{p}_i^{t-1}\left(\boldsymbol{\ell}_i^t\ln\frac{1}{\beta}\right)
\end{aligned}$$

Since $\sum_i\mathbf{p}_i^{t-1}\boldsymbol{\ell}_i^t$ is the expected local cost, we have achieved our goal. ■

5.2 Running only one algorithm

The problem becomes more intricate when we can run only one of the n algorithms at a time. Such may be the case, for example, if we are combining several paging algorithms but the system cannot afford the time required to simulate all of the algorithms in order to maintain their losses.

This is a version of the Bandits problem studied by Auer, Cesa-Bianchi, Freund, and Schapire [ACBFS95, ACBFS98]. Bandits is a variant of Experts, where each time step the algorithm can see the loss of only the expert chosen. (The problem's name derives from slot machines.) Auer *et al.* show that, by mixing the Hedge distribution appropriately with the uniform distribution, they can guarantee a loss of at most $O(\sqrt{Tn\log n})$ more than the best expert's loss, where T is the number of time steps.

To mesh better with the phrasing of Auer *et al.*'s, we consider the scenario where each time step every expert incurs a *reward* in [0,1], and we wish to maximize our *gain*. Our scenario adds to theirs the concept of a switching cost d, which works as follows: In time round t, expert i has a *true gain* $\mathbf{x}_j^t$ in $[0,1]$, but the gain the algorithm actually sees is an approximation to this called the *observed gain* $\widetilde{\mathbf{x}}_j^t$ (also in $[0,1]$). The true gain and the observed gain are related in that, if the algorithm remains at a single expert from t_0 to t_1, then the total observed gain $\sum_{t=t_0}^{t_1}\widetilde{\mathbf{x}}_j^t$ is at most d less than the total actual gain $\sum_{t=t_0}^{t_1}\mathbf{x}_j^t$. (This somewhat convoluted way of incorporating the switching cost comes from the paging case in Example 5.1. When we switch from one algorithm to another, we do not know the actual cost incurred by the new algorithm, since we have not kept track of where it is. Our model assumes that all the algorithms have the property that, regardless of the request sequence, the initial cache cannot affect the total cost by more than d.)

This switching cost removes the luxury (which Auer *et al.* enjoy) of choosing an expert independently each time round, because switching as often as this implies is quite expensive. One possible solution to this problem, which we pursue, is to divide time into segments of s steps. (We choose s later.) We choose independently from the distribution at the beginning of each time segment, and we stay there for the duration of the segment. Behaving in this way is equivalent to running Auer *et al.*'s algorithm for $\frac{T}{s}$ time steps, where in each step an expert's maximum loss is at most s, rather than only 1.

Algorithm Hedge-Bandit The algorithm has two parameters, s and β. For each time segment of s steps, the algorithm does the following.

1. We choose one expert $\mathbf{i}^t$ for the time segment t (time steps ts through $(t+1)s$) based on the probabilities
$$\mathbf{p}_j^{t-1} = (1-\gamma)\widehat{\mathbf{p}}_j^{t-1} + \frac{\gamma}{n}\,,$$
where $\widehat{\mathbf{p}}^{t-1}$ is the probability distribution used by Hedge.
2. We observe the gain $\widetilde{\mathbf{x}}_{\mathbf{i}^t}^t$ for the segment. (For $j \neq \mathbf{i}^t$, we take $\widetilde{\mathbf{x}}_j^t$ to be 0.)
3. We let $\widehat{\mathbf{x}}_j^t = \widetilde{\mathbf{x}}_j^t / \mathbf{p}_j^{t-1}$, and give this vector $\widehat{\mathbf{x}}^t$ to Hedge in order to compute $\widehat{\mathbf{p}}^t$ for the next time segment.

Analyzing this algorithm requires the following theorem of Auer *et al.* generalizing the bound on Hedge's performance (Theorem 3.2) to the case when an expert's gain may be as much as M per time step.

Theorem 5.2 *If each of a set of n experts experiences a sequence $\mathbf{x}_j$ of gains in $[0, M]$, then Hedge configured with $\beta \in (0,1)$ has expected gain $\sum_t \sum_j \mathbf{p}_j^t \mathbf{x}_j^t$ of at least*

$$\sum_{t=1}^{T} \mathbf{x}_k^t - \frac{\ln n}{\ln \beta} - \frac{\beta^M - 1 - M \ln \beta}{M^2 \ln \beta} \sum_{t=1}^{T} \sum_{j=1}^{n} \mathbf{p}_j^t \left(\mathbf{x}_j^t\right)^2 \,,$$

for all experts k.

We use this in the following theorem bounding the performance of Hedge-Bandit.

Theorem 5.3 *The expected gain of Hedge-Bandit is at least*

$$G - (1-\gamma)\frac{T}{s}d - \frac{1-\gamma}{\gamma} sn \ln n - (e-1)\gamma G\,,$$

where G is the largest total actual gain acquired by any single expert, and where $\beta = e^{\gamma/sn}$.

By choosing appropriate values for γ and s as described in the following corollary, we bound our gain relative to the best of the algorithms.

Corollary 5.4 *The expected gain of Hedge-Bandit is at least*

$$G - 3.6\sqrt[3]{dnT^2 \ln n}\,,$$

where G is the largest total actual gain acquired by any single expert, if we choose the parameters

$$\gamma = 0.7\sqrt[3]{\frac{dn \ln n}{T}},\; s = 0.8\sqrt[3]{\frac{Td^2}{n \ln n}}\,.$$

The proof of Theorem 5.3 closely follows the technique used by Auer *et al.* [ACBFS98].

Proof of Theorem 5.3. Let k be the expert acquiring the largest total actual gain. Because $\mathbf{p}_j^t \geq \frac{\gamma}{n}$ for any expert j in any time segment t, the scaled observed gain $\widehat{\mathbf{x}}_j^t = \widetilde{\mathbf{x}}_j^t / \mathbf{p}_j^t$ is at most sn/γ. So we take M to be sn/γ (and recall $\beta = e^{\gamma/sn}$) in applying Theorem 5.2 for the following bound.

$$\begin{aligned}
\sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \widehat{\mathbf{x}}_j^t &\geq \sum_{t=1}^{T/s} \widehat{\mathbf{x}}_k^t - \frac{\ln n}{\ln \beta} - \frac{\beta^M - 1 - M\ln\beta}{M^2 \ln\beta} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \left(\widehat{\mathbf{x}}_j^t\right)^2 \\
&= \sum_{t=1}^{T/s} \widehat{\mathbf{x}}_k^t - \frac{sn\ln n}{\gamma} - \frac{(e-2)\gamma}{sn} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \left(\widehat{\mathbf{x}}_j^t\right)^2 \qquad (5.1)
\end{aligned}$$

Now, because $\mathbf{p}_j^t \geq (1-\gamma)\widehat{\mathbf{p}}_j^t$, we can observe the following.

$$\begin{aligned}
\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \widehat{\mathbf{x}}_j^t &= \widehat{\mathbf{p}}_{\mathbf{i}^t}^t \frac{\widetilde{\mathbf{x}}_{\mathbf{i}^t}^t}{\mathbf{p}_{\mathbf{i}^t}^t} \leq \frac{\widetilde{\mathbf{x}}_{\mathbf{i}^t}^t}{1-\gamma} \\
\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \left(\widehat{\mathbf{x}}_j^t\right)^2 &= \widehat{\mathbf{p}}_{\mathbf{i}^t}^t \frac{\widetilde{\mathbf{x}}_{\mathbf{i}^t}^t}{\mathbf{p}_{\mathbf{i}^t}^t} \widehat{\mathbf{x}}_{\mathbf{i}^t}^t \leq \frac{s}{1-\gamma} \widehat{\mathbf{x}}_{\mathbf{i}^t}^t = \frac{s}{1-\gamma} \sum_{j=1}^{n} \widehat{\mathbf{x}}_j^t
\end{aligned}$$

We use both of these facts, along with (5.1) and the relationship of the observed gains $\widetilde{\mathbf{x}}$ to the actual gains $\mathbf{x}$, to bound the total gain, $\sum_{t=1}^{T/s} \mathbf{x}_{\mathbf{i}^t}^t$.

$$\begin{aligned}
\sum_{t=1}^{T/s} \mathbf{x}_{\mathbf{i}^t}^t &\geq \sum_{t=1}^{T/s} \widetilde{\mathbf{x}}_{\mathbf{i}^t}^t \geq (1-\gamma) \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \widehat{\mathbf{x}}_j^t \\
&\geq (1-\gamma)\sum_{t=1}^{T/s} \widehat{\mathbf{x}}_k^t - \frac{1-\gamma}{\gamma} sn\ln n - \frac{(e-2)(1-\gamma)\gamma}{sn} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{p}}_j^t \left(\widehat{\mathbf{x}}_j^t\right)^2 \\
&\geq (1-\gamma)\sum_{t=1}^{T/s} \widehat{\mathbf{x}}_k^t - \frac{1-\gamma}{\gamma} sn\ln n - \frac{(e-2)\gamma}{n} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widehat{\mathbf{x}}_j^t \qquad (5.2)
\end{aligned}$$

To get the expected gain $\mathbf{E}\left[\sum_t \mathbf{x}_{\mathbf{i}^t}^t\right]$, we first observe that $\mathbf{E}\left[\widehat{\mathbf{x}}_j^t\right]$ equals $\mathbf{E}\left[\mathbf{x}_j^t\right]$:

$$\begin{aligned}
\mathbf{E}\left[\widehat{\mathbf{x}}_j^t\right] &= \mathbf{E}_{\mathbf{i}^1,\ldots,\mathbf{i}^{t-1}}\left[\mathbf{E}_{\mathbf{i}^t}\left[\widehat{\mathbf{x}}_j^t \mid \mathbf{i}^1,\ldots,\mathbf{i}^{t-1}\right]\right] \\
&= \mathbf{E}_{\mathbf{i}^1,\ldots,\mathbf{i}^{t-1}}\left[\mathbf{E}_{\mathbf{i}^t}\left[\mathbf{p}_j^t \cdot \frac{\widetilde{\mathbf{x}}_j^t}{\mathbf{p}_j^t} + (1-\mathbf{p}_j^t)\cdot 0\right]\right] \\
&= \mathbf{E}_{\mathbf{i}^1,\ldots,\mathbf{i}^{t-1}}\left[\mathbf{E}_{\mathbf{i}^t}\left[\widetilde{\mathbf{x}}_j^t\right]\right] = \widetilde{\mathbf{x}}_j^t\,.
\end{aligned}$$

We continue from (5.2), using the fact that the observed gain $\widetilde{\mathbf{x}}_j^t$ is between the actual gain $\mathbf{x}_j^t$ and $\mathbf{x}_j^t - d$.

$$\begin{aligned}
\mathbf{E}\left[\sum_{t=1}^{T/s} \mathbf{x}_{\mathbf{i}^t}^t\right] &\geq (1-\gamma)\sum_{t=1}^{T/s} \widetilde{\mathbf{x}}_k^t - \frac{1-\gamma}{\gamma} sn\ln n - \frac{(e-2)\gamma}{n} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \widetilde{\mathbf{x}}_j^t \\
&\geq (1-\gamma)\sum_{t=1}^{T/s} (\mathbf{x}_k^t - d) - \frac{1-\gamma}{\gamma} sn\ln n - \frac{(e-2)\gamma}{n} \sum_{t=1}^{T/s}\sum_{j=1}^{n} \mathbf{x}_j^t \\
&\geq (1-\gamma)G - (1-\gamma)\frac{T}{s}d - \frac{1-\gamma}{\gamma} sn\ln n - (e-2)\gamma G
\end{aligned}$$

Chapter 6

Relating MTS and Experts

A second way of extending the results of Section 3.4 is to consider the converse question: How do MTS algorithms perform on the Experts problem? Besides the academic and historic interest in such a question, the work-function approach used in metrical task systems — a very different approach from the multiplicative weight-updating technique studied for Experts up to now — may prove more useful in some learning situations.

In this chapter, we first look at a generic theorem translating any r-unfair MTS algorithm into an Experts algorithm. Then we illustrate an analysis of one particular MTS-derived algorithm (Linear on two points/experts) in the Experts problem. And finally we look at a small empirical comparison of how our large set of Experts/MTS algorithms performs on real data inspired by process migration.

6.1 General relation

As Section 3.4 illustrates, achieving an unfair competitive ratio for the uniform MTS problem is similar to achieving a partitioning bound in the Experts setting. The parameter r allows us to trade off the L_P and k_P coefficients, similarly to β in Thresh and Share.

Conversion from MTS to Experts

The following theorem makes the relationship formal.

Theorem 6.1 *Let A be a randomized algorithm for the MTS problem on the n-point uniform space that, given r, achieves an r-unfair competitive ratio of $\rho_{n,r}$. Then this implies an algorithm A' for the Experts setting has expected loss at most*

$$\frac{\rho_{n,r}}{r} L_P + \rho_{n,r} k_P + b \,,$$

for any partition P, for some constant b that may depend on r and n (typically, $b \leq r$).

Remark. Note that if $\rho_{n,r} = r + \log n$ and $\varepsilon = \frac{1}{r} \log n$, then this partitioning bound translates to $(1+\varepsilon)L_P + (1+\frac{1}{\varepsilon})k_P \log n$, analogous to the bound that Thresh and Share achieve (Theorems 3.3 and 3.4).

Proof. At each time step, our algorithm A' uses whatever distribution A currently has. When it receives loss vector $\boldsymbol{\ell}^t$, it gives a scaled version $\frac{1}{r}\boldsymbol{\ell}^t$ to A so that A can modify its distribution for A' to use in the next time step.

Consider any sequence of loss vectors $\boldsymbol{\ell}$ and any partition P. Let $\mathbf{p}^t$ represent the probability vector on states that A uses for the tth time step (and which A' uses for the $(t+1)$st time step). So, given a loss vector $\boldsymbol{\ell}$, A' has expected loss $\mathbf{p}^{t-1} \cdot \boldsymbol{\ell}^t$. But A "believes" it is paying $d(\mathbf{p}^{t-1}, \mathbf{p}^t)$ for movement and $\mathbf{p}^t \cdot (\frac{1}{r}\boldsymbol{\ell}^t)$ for processing. (Because we use an r-unfair ratio, in another sense A believes it pays $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$ for processing while its adversary pays only $\mathbf{p}^t \cdot (\frac{1}{r}\boldsymbol{\ell}^t)$.)

We will show that the expected loss to A is at most

$$\begin{aligned} \mathbf{E}[loss_A] &\leq \sum_t \left(d(\mathbf{p}^{t-1}, \mathbf{p}^t) + \mathbf{p}^t \cdot \boldsymbol{\ell}^t\right) \\ &= \mathbf{E}\left[move(\mathbf{v}_A) + r\ local(\tfrac{1}{r}\boldsymbol{\ell}, \mathbf{v}_A)\right] \ . \end{aligned} \tag{6.1}$$

Once we have this, we can let $\mathbf{v}$ be the action sequence corresponding to partition P. This sequence remains at a single expert within each interval of P, so that $move(\mathbf{v}) \leq k_P$ and $local(\frac{1}{r}\boldsymbol{\ell}, \mathbf{v}) = \frac{1}{r}L_P$. Continuing from (6.1), because A has r-unfair ratio ρ, the expected loss is at most

$$\rho_{n,r}\left(move(\mathbf{v}) + local(\tfrac{1}{r}\boldsymbol{\ell}, \mathbf{v})\right) + b \leq \rho_{n,r}\left(k_P + \frac{1}{r}L_P\right) + b\ ,$$

as the theorem states.

To show (6.1), consider a specific trial $\boldsymbol{\ell}^t$. The expected loss to A is $\mathbf{p}^{t-1} \cdot \boldsymbol{\ell}^t$. We bound this by $d(\mathbf{p}^{t-1}, \mathbf{p}^t) + \mathbf{p}^t \cdot \boldsymbol{\ell}^t$, and (6.1) follows.

$$\begin{aligned} \sum_i \mathbf{p}_i^{t-1}\boldsymbol{\ell}_i^t &= \sum_i \left(\mathbf{p}_i^{t-1} - \mathbf{p}_i^t\right)\boldsymbol{\ell}_i^t + \sum_i \mathbf{p}_i^t\boldsymbol{\ell}_i^t \\ &\leq \sum_{i:\mathbf{p}_i^{t-1} > \mathbf{p}_i^t} \left(\mathbf{p}_i^{t-1} - \mathbf{p}_i^t\right)\boldsymbol{\ell}_i^t + \sum_i \mathbf{p}_i^t\boldsymbol{\ell}_i^t \\ &\leq \sum_{i:\mathbf{p}_i^{t-1} > \mathbf{p}_i^t} \left(\mathbf{p}_i^{t-1} - \mathbf{p}_i^t\right) + \sum_i \mathbf{p}_i^t\boldsymbol{\ell}_i^t \\ &= d(\mathbf{p}^{t-1}, \mathbf{p}^t) + \mathbf{p}^t \cdot \boldsymbol{\ell}^t \end{aligned}$$

The next-to-last step follows because loss vectors are bounded by 1. ■

Corollaries to our conversion

This theorem immediately results in new Experts algorithms with approaches very different from established multiplicative-update algorithms like Thresh and Share. The first comes from applying Theorem 6.1 to our unfair analysis of Marking (Theorem 2.5).

Corollary 6.2 *For the Experts problem, Marking has a partitioning bound of at most*

$$(1+\varepsilon)\, H_n\, L_P + \left(1 + \frac{1}{\varepsilon}\right) H_n\, k_P + H_n\ ,$$

where $\varepsilon = \frac{1}{r}$.

Because the L_P coefficient here approaches H_n, this bound is much worse than the bound provided by the multiplicative-update algorithms (where the L_P coefficient approaches 1).

But if we instead use our r-unfair analysis of Odd-Exponent (Theorem 4.5), we get a bound comparable to that of Thresh and Share.

Corollary 6.3 *For the Experts problem, if we choose $t = \ln n$, then Odd-Exponent has a partitioning bound of at most*

$$(1+\varepsilon)L_P + \left(1 + \frac{1}{\varepsilon}\right) 2e \ln n\, k_P + \frac{2e}{\varepsilon} + 2\,,$$

where $\varepsilon = \frac{2e}{r} \ln n$.

This is very comparable to the Share bound; the difference is that the k_P coefficient is about $2e$ times what Share achieves.

At least some of this $2e$ factor is likely an artifact of our analysis. Based on the $t = 1$ case (Theorem 4.2), we might suppose that the $2n^{1/t}t$ term of Theorem 4.5 is twice the possible guarantee. But also, using Theorem 6.1 to convert the MTS unfair competitive ratio to an Experts partitioning bound can involve some loss. This is illustrated by our direct analysis of Linear on two experts.

6.2 Direct analysis of Linear

Of course, we can analyze an algorithm directly in the Experts environment rather than use Theorem 6.1. We illustrate this with the Linear algorithm on two experts.

To review: The Linear algorithm on two points maintains the work function $\mathbf{OPT}_1$ and $\mathbf{OPT}_2$ for the two points and allocates probability

$$\mathbf{p}_1 = \frac{1}{2} + \frac{\mathbf{OPT}_2 - \mathbf{OPT}_1}{2}$$

to the first point and the remainder to the second. That is, Linear moves probability linearly between experts, so that an expert's probability is zero when it is pinned. This strategy is optimal for the two-point unfair MTS problem, achieving a ratio of $r + 1$ (Theorem 4.2).

Before we analyze Linear in the Experts problem, notice that if we use Theorem 6.1 on the r-unfair analysis in Theorem 4.2, we get the following.

Corollary 6.4 *For the Experts problem with two experts, Odd-Exponent has a partitioning bound of at most*

$$(1+\varepsilon)L_P + \left(1 + \frac{1}{\varepsilon}\right) k_P + \frac{1}{2\varepsilon} + \frac{1}{2}\,,$$

where $\varepsilon = \frac{1}{r}$.

We now analyze Linear directly; this analysis effectively halves the k_P coefficient.

Theorem 6.5 *For the Experts problem, the partitioning bound of Linear is at most*

$$(1+\varepsilon)\, L_P + \left(1 + \frac{1}{\varepsilon}\right) \frac{1}{2} k_P\,,$$

where $\varepsilon = \frac{1}{2r}$, provided r is an integer.

Proof. Consider segment i of the partition with loss $\mathbf{L}^i$. Assume without loss of generality that the better expert for the segment is expert 1. (So $\mathbf{L}^i$ represents the total loss to expert 1 in the segment.) Let δ represent the fractional component of $\mathbf{OPT}_2 - \mathbf{OPT}_1$ (that is, $\delta = (\mathbf{OPT}_2 - \mathbf{OPT}_1) - \lfloor \mathbf{OPT}_2 - \mathbf{OPT}_1 \rfloor$). (If we can assume the losses are always either 0 or 1, then the proof can be simplified by ignoring δ (it is always 0) and ignoring cases 2 and 4 below (which occur only when p_1 or p_2 is 0).)

We will use a potential function over this segment of

$$\Phi = rp_2^2 + \frac{1}{2}p_2 + \frac{\delta(1-\delta)}{4r} .$$

Notice that Φ is always between 0 and $r + \frac{1}{2}$. (If $\mathbf{OPT}_2 - \mathbf{OPT}_1 = -r + \delta$ for $\delta \in [0, 1]$, then $p_2 = 1 - \frac{\delta}{2r}$ and so $\Phi = r + \frac{1}{2} - \delta$.)

Say the algorithm receives loss vector $\langle \ell_1, \ell_2 \rangle$. Our goal is to show that the algorithm's cost plus potential change is at most $\ell_1(1 + \frac{1}{2r})$. If we know this, then the total cost for segment i is at most $(1 + \frac{1}{2r})\mathbf{L}^i$ plus the maximum potential change between segments, $r + \frac{1}{2}$. Thus the total cost for the partition is at most

$$\sum_{i=1}^{k} \left(\left(1 + \frac{1}{2r}\right) \mathbf{L}^i + r + \frac{1}{2} \right) = \left(1 + \frac{1}{2r}\right) L_P + \left(r + \frac{1}{2}\right) k_P .$$

We can assume that $\langle \ell_1, \ell_2 \rangle$ is 0 in one of its components for the following reason. Let $\hat{\ell} = \min\{\ell_1, \ell_2\}$ and divide the vector into two pieces $\left\langle \hat{\ell}, \hat{\ell} \right\rangle$ and $\left\langle \ell_1 - \hat{\ell}, \ell_2 - \hat{\ell} \right\rangle$. On the first piece the algorithm's cost is $\hat{\ell}$ with no effect on probability or potential; and on the second the cost is (as we will show) at most $(\ell_1 - \hat{\ell})(1 + \frac{1}{2r})$. So for both pieces the total cost plus potential change is at most $\hat{\ell} + (\ell_1 - \hat{\ell})(1 + \frac{1}{2r}) \le \ell_1(1 + \frac{1}{2r})$. We split the remaining possibilities into four cases.

Case 1: The vector is $\langle \ell, 0 \rangle$ and $\mathbf{OPT}_2 - \mathbf{OPT}_1 \ge -r + \ell$. Then $\mathbf{OPT}_2 - \mathbf{OPT}_1$ increases by ℓ and so p_1 loses $\ell/2r$ probability to p_2. Notice that the last term of the potential function increases most when δ is initially 0. The amortized cost, then, is

$$p_1\ell + \Delta\Phi \le p_1\ell + \left(p_2\ell + \tfrac{\ell^2}{4r} + \tfrac{\ell}{4r} + \tfrac{\ell(1-\ell)}{4r}\right) = \ell\left(1 + \tfrac{1}{2r}\right) .$$

Case 2: The vector is $\langle \ell, 0 \rangle$ and for some $\tilde{\ell} \in [0, \ell)$ we have $\mathbf{OPT}_2 - \mathbf{OPT}_1 = -r + \tilde{\ell}$. Then p_2 increases from $1 - \tilde{\ell}/2r$ to 1, and δ drops from $\tilde{\ell}$ to 0. The amortized cost is

$$p_1\ell + \Delta\Phi = \tfrac{\tilde{\ell}}{2r}\ell + \left(\tilde{\ell} - \tfrac{\tilde{\ell}^2}{4r} + \tfrac{\tilde{\ell}}{4r} - \tfrac{\tilde{\ell}(1-\tilde{\ell})}{4r}\right) \le \ell\left(1 + \tfrac{1}{2r}\right) .$$

Case 3: The vector is $\langle 0, \ell \rangle$ and $\mathbf{OPT}_2 - \mathbf{OPT}_1 \le r - \ell$. Then p_2 loses $\ell/2r$ probability to p_1. The last term of the potential function increases by at most $\ell(1-\ell)/4r$. The amortized cost is

$$p_2\ell + \Delta\Phi \le p_2\ell + \left(-p_2\ell + \tfrac{\ell^2}{4r} - \tfrac{\ell}{4r} + \tfrac{\ell(1-\ell)}{4r}\right) = 0 .$$

Case 4: The vector is $\langle 0, \ell \rangle$ and for some $\tilde{\ell} \in [0, \ell)$ we have $\mathbf{OPT}_2 - \mathbf{OPT}_1 = r - \tilde{\ell}$. Then p_2 drops from $\tilde{\ell}/2r$ to 0, and, because r is integral, δ drops from $1 - \tilde{\ell}$ to 0. The amortized cost is

$$p_2\ell + \Delta\Phi = \tfrac{\tilde{\ell}}{2r}\ell + \left(-\tfrac{\tilde{\ell}^2}{4r} - \tfrac{\tilde{\ell}}{4r} - \tfrac{(1-\tilde{\ell})\tilde{\ell}}{4r}\right) \le 0 .$$

In all cases, the algorithm's cost is at most $\ell_1(1 + \frac{1}{2r})$. ∎

6.3 Process migration experiments

We now examine some brief experimental results comparing several algorithms, including many Experts/MTS algorithms, on data representing a process migration problem. Process migration has aspects of both the MTS problem and the Experts settings. There is a cost to move between machines, but there is also zero lookahead.

For process migration data, we collected load averages collected from 112 machines around the CMU campus. We queried each machine every five minutes for 6.5 days. From these machines, we selected 32 that were busy enough to be interesting for this analysis.

Each five-minute interval corresponds to a trial with loss vector $\boldsymbol{\ell}^t$. For machine i, we set $\boldsymbol{\ell}_i^t = 1$ if the machine had a large load average (more than 0.5), and $\boldsymbol{\ell}_i^t = 0$ if it had a small load average. The intent of this is to model the decision faced by a "user-friendly" background process that suspends its work if someone else is using the same machine.

We took the distance between the machines to be 0.1, indicating that 30 seconds of computation would be lost for movement between machines. In research process migration systems, the time for a process to move is roughly proportional to its size. For a 100-KB process, the time is about a second [Esk90]. Our distance corresponds to large but reasonable memory usage.

Our simulations compared the performance of nine algorithms, including four simple control algorithms:

Uniform The algorithm picks a random machine and stays there for all trials.

Greedy After each trial the algorithm moves to the machine that incurred the least loss in that trial (with ties broken randomly).

Least-Used After each trial the algorithm moves to the machine that has incurred the least total loss so far.

Recent The algorithm moves to the machine that has incurred the least loss over the last k trials.

We implemented Work-Function, Marking, Odd-Exponent (with $t = 3$), Thresh, and Share. (Efficiently implementing Odd-Exponent to compensate for Assumption 4.1 is a challenge; we discuss this at the end of this section.

Because these algorithms have tunable parameters, we divided the data into a training set and a test set, 936 trials each. We optimized parameters on the training set and report the performance with these parameters on the test set. We also present the performance of each algorithm with a "naive" parameter setting, to give a sense of the dependence of the behavior of the algorithm on the tuning of its parameters.

For each algorithm we determined the expected loss for the probability vectors they calculated. One valid criticism of using probabilistic algorithms in practice is the variance between runs; so we also calculated the standard deviation over 200 trials of each algorithm. To get a feel of how each algorithm behaves, we finally computed the expected number of moves.

This data is summarized in Table 6.1 where costs are given relative to the optimal off-line sequence, which suffered a loss of 3.8 and moved 8 times in the test sequence.

We also tried an inter-machine distance of 1.0. Table 6.2 summarizes these results. For an inter-machine distance of 1.0, the optimal off-line sequence suffered a loss of 11 and moved 6 times during the 936 trials. (As one would expect, the loss is higher but there are fewer movements.)

Comparing these algorithms to the simpler control algorithms indicates that their added sophistication does indeed help. The numbers seem to indicate that the MTS-based algorithms are less sensitive to parameter settings. The specific experiments summarized here show that the MTS algorithms performing somewhat better; if the parameters are set based on the *test* data, this difference decreases.

algorithm	parameter setting	cost ratio	std dev	expected moves	naive setting	cost ratio
Uniform		206.69	29.03	0.00		
Greedy		55.11	4.33	265.34		
Least-Used		117.71	0.00	5.00		
Recent	$k : 6$	17.92	0.00	103.00	$k : 5$	24.37
Work-Function	$r : 1.0$	5.66	0.00	17.00	$r : 1.0$	5.66
Marking	$r : 1.0$	5.97	0.72	20.54	$r : 1.0$	5.97
Odd-Exponent	$t : 3, r : 10.0$	5.96	0.79	15.84	$t : 3, r : 1.0$	6.05
Thresh	$\beta : 9.5 \times 10^{-6}, \alpha : 10^{-4}$	7.16	0.66	14.53	$\beta : 0.5, \alpha : 0.01$	20.89
Share	$\beta : 5.2 \times 10^{-7}, \alpha : 10^{-8}$	6.55	0.63	14.58	$\beta : 0.5, \alpha : 0.01$	19.44

Table 6.1: Performance relative to optimal off-line sequence ($d = 0.1$) on process migration data.

algorithm	parameter setting	cost ratio	std dev	expected moves	naive setting	cost ratio
Uniform		71.40	10.90	0.00		
Greedy		40.75	2.91	265.34		
Least-Used		41.07	0.00	5.00		
Recent	$k : 11$	6.62	0.00	41.00	$k : 5$	19.71
Work-Function	$r : 1.0$	3.34	0.00	13.00	$r : 1.0$	3.34
Marking	$r : 0.4$	3.74	0.40	20.54	$r : 1.0$	4.27
Odd-Exponent	$t : 3, r : 1.0$	3.36	0.51	15.84	$t : 3, r : 1.0$	3.36
Thresh	$\beta : 0.027, \alpha : 10^{-8}$	5.52	0.34	10.66	$\beta : 0.5, \alpha : 0.01$	8.20
Share	$\beta : 0.044, \alpha : 10^{-8}$	5.59	0.39	11.56	$\beta : 0.5, \alpha : 0.01$	7.68

Table 6.2: Performance relative to optimal off-line sequence ($d = 1.0$) on process migration data.

algorithm	competitive ratio	partitioning bound
Two-Region ($n = 2$)	$r + 1$ (Th 4.2)	$(1 + \varepsilon)\, L_P + \left(1 + \frac{1}{\varepsilon}\right) \frac{1}{2} k_P$ (Th 6.5)
Marking	$(r + 1)H_n$ (Th 2.5)	$(1 + \varepsilon) H_n L_P + \left(1 + \frac{1}{\varepsilon}\right) H_n\, k_P$ (Cor 6.2)
Odd-Exponent	$r + 2e \ln n$ (Th 4.5)	$(1 + \varepsilon)\, L_P + \left(1 + \frac{1}{\varepsilon}\right) 2e \ln n\, k_P$ (Cor 6.3)
Thresh	unbounded	$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) L_P + \left(\frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)}\right) k_P$ (Th 3.3)
Share	$r + 6.4 \ln\left(n(r + 1)\right) + 4$ (Th 3.5)	$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) L_P + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right) k_P$ (Th 3.4)

Table 6.3: Summary of theoretical results.

The numbers indicate that Work-Function slightly outperforms the randomized algorithms, despite its worse theoretical guarantee. This is not too surprising because a randomized algorithm is essentially using its probability distribution to hedge its bets, placing probability on states that do not necessarily appear optimal. This is somewhat analogous to a stock market, in which the main reason to diversify is to minimize the downside risk more than to maximize expected gain. In these experiments, all the algorithms performed better than their worst-case guarantees. In practice, Odd-Exponent follows Work-Function very closely, although it smooths the transitions between states.

Implementing Odd-Exponent

In an implementation of Odd-Exponent, using $\mathbf{OPT}$ values strictly as defined introduces a problem: The algorithm could allocate negative probability to an expert. (Consider the case where expert 1 has $\mathbf{OPT}_1 = r$ while the rest are at zero.) The analysis of Theorem 4.8 skirts the issue by assuming Assumption 4.1.

If we wish to implement Odd-Exponent, we must confront the possibility that tasks observed will not obey this condition. We can address this by using a modification of the work function, $\widetilde{\mathbf{OPT}}$, in computing the probability distribution of the strategy. This $\widetilde{\mathbf{OPT}}$ is computed as follows. Say the strategy receives a loss vector ℓ. We will change $\widetilde{\mathbf{OPT}}_i$ to become, not $\min\{\widetilde{\mathbf{OPT}}_i + \ell_i, \min_j \widetilde{\mathbf{OPT}}_j + \ell_j + r\}$ as for the work function, but $\min\{\widetilde{\mathbf{OPT}}_i + \ell_i, x\}$, where x is the greatest value such that no probabilities are negative. (In an implementation one can compute x by considering the function returning the minimum probability for a given x and using numerical techniques to find where this function reaches zero.) This avoids negative probabilities because each probability that would have become negative with the unmodified work function becomes zero instead.

This modification maintains the same competitive ratio because we can think of it as dividing each cost vector into two pieces, $\tilde{\ell}$ and $\ell - \tilde{\ell}$, where $\tilde{\ell} = \widetilde{\mathbf{OPT}}^{t+1} - \widetilde{\mathbf{OPT}}^{t}$. For $\tilde{\ell}$, the algorithm is competitive with respect to the off-line player's cost on $\tilde{\ell}$ (which itself is less than the off-line player's cost on ℓ). For $\ell - \tilde{\ell}$, the algorithm will pay nothing, since the vector is nonzero only at states where $\widetilde{\mathbf{OPT}} = x$, and these states have no probability.

Chapter 7

The unfair paging problem

One of the strands running beneath this thesis is the usefulness of the notion of *unfairness* in on-line analysis. This is most apparent in our development of a $polylog(n)$ MTS algorithm, but the machine-learning notion of a partitioning bound (in the related but different Experts problem) is also actually a question of unfairness. What unfairness allows us to do is to build more sophisticated bounds than a straight competitive ratio allows, essentially by parameterizing the relative importance of different costs. This prevents an algorithm from ignoring one part of the costs. For example, standard algorithms for the MTS algorithm can be sloppy with local costs as long as they are only a constant factor more than the movement cost. Adding unfairness to the model forces us to be careful with both aspects.

One can naturally ask if this advantage can be extended to other problems. In this chapter, we see that it can, in particular to the Paging problem.

Problem Paging An on-line algorithm controls a cache of k pages and sees a sequence of memory requests $\boldsymbol{\rho}^1, \boldsymbol{\rho}^2, \ldots$. When an item outside the current cache is requested, the algorithm incurs a **page fault** and must load the requested page into the cache, evicting some other page of its choice. The goal of the algorithm is to minimize the number of page faults.

Fiat *et al.* describe Marking, a randomized algorithm for Paging (similar to the eponymous MTS algorithm by Borodin, Linial, and Saks), with a competitive ratio of $O(\log k)$ [FKL$^+$91, BLS92]. (Fiat *et al.* also show that every Paging algorithm must have a competitive ratio of at least $\Omega(\log n)$.)

Algorithm Marking ([FKL$^+$91]) For each of the k cache locations, we have space for a mark, initially empty. When a page in the cache is requested, we mark its location. When a page outside the cache is requested, we pick a random unmarked location, eject its page, and mark the location. If all locations are marked, we clear the marks and begin a new phase.

Theorem 7.1 ([FKL$^+$91]) *Marking has a competitive ratio of $2H_k$ for Paging.*

How to incorporate unfairness into Paging is not obvious. Our approach is the following: Suppose that on a page fault, the *off-line* algorithm is allowed the additional power to "rent" the requested page at a cost of only $\frac{1}{r}$ (think of $r = \log k$), compared with the cost of 1 for actually loading the page into the cache. Renting means that the memory request is serviced but the requested page is *not* brought into the cache and the off-line cache is *not* modified. So, for instance, if the off-line algorithm rents a page and then the same page is requested again, the off-line algorithm incurs another page fault. The *on-line* algorithm has no such privilege. (Technically, it is convenient to allow the on-line algorithm to rent for a cost of 1; at best, this helps the on-line algorithm by a factor of two.) The question we examine is, what competitive ratio can be achieved in this scenario? This question can be thought of as the unfair version of Paging, because we have split the cost into renting and loading, with the off-line algorithm having an unfair advantage on renting.

For this harder unfair problem, no algorithm can achieve an r-unfair competitive ratio less than r (consider a sequence where each request is to a new page), nor can any algorithm achieve a competitive ratio less than $O(\log k)$. Marking achieves competitive ratio $O(r \log k)$. We consider the question of whether one can achieve ratio $O(r + \log k)$. The main result of this paper is that we can, using Hedge together with a notion of phases similar to Marking.

7.1 Motivation

Because the problem stated above is not obviously self-motivating, we begin by presenting two motivations, one from paging and another from the k-server problem.

Finely-competitive paging Request sequences in practice often consist of a core working set of frequently requested pages, together with occasional assorted memory requests, where this working set slowly changes over time. Suppose that, in hindsight, the request sequence can be partitioned into time periods containing working sets $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^m$ respectively, where within each time period the number of requests to pages *outside* the current working set is $\mathbf{o}^1, \mathbf{o}^2, \ldots, \mathbf{o}^m$. Furthermore, suppose that each working set is small enough to fit within the memory cache ($|\mathbf{W}^i| \leq k$). In this scenario, one off-line strategy in our "unfair" model is to load the current working set into the cache and to rent the requests outside the current working set, at a cost of

$$\frac{1}{r}(\mathbf{o}^1 + \cdots + \mathbf{o}^m) + |\mathbf{W}^1| + |\mathbf{W}^2 \setminus \mathbf{W}^1| + \cdots + |\mathbf{W}^m \setminus \mathbf{W}^{m-1}|\,.$$

Taking $r = \log k$, an algorithm with unfair competitive ratio $O(r + \log k)$ must pay at most $O(\log k)$ times this, or

$$O\left((\mathbf{o}^1 + \cdots + \mathbf{o}^m) + (\log k)(|\mathbf{W}^1| + |\mathbf{W}^2 \setminus \mathbf{W}^1| + \cdots + |\mathbf{W}^m \setminus \mathbf{W}^{m-1}|)\right)\,.$$

So, if the sequence involves only a few working sets or if their differences are small compared to the $\mathbf{o}^i$, the on-line algorithm is only a small (constant) factor from the optimal service sequence.

Here is a simple concrete example. Suppose that the request sequence repeatedly cycles over a fixed set of $k + 1$ pages. In that case, the deterministic LRU algorithm has competitive ratio k (it faults on every request) and Marking has competitive ratio $O(\log k)$ (in expectation, it makes $O(\log k)$ page faults per cycle). However, our algorithm in this case is required to have an $O(1)$ ratio because we can view this sequence as having a single fixed working set of size k, with one additional request per cycle. In other words, in the unfair model, the off-line algorithm could simply incur a cost of $\frac{1}{r} = \frac{1}{\log k}$ per cycle by renting.

In a sense, this goal can be viewed as follows. The motivation of the competitive ratio measure itself is to allow the on-line algorithm to perform worse on "harder" sequences but to require it to perform better on "easier" ones. Unfairness provides a more fine-grained measure, in which we split the off-line cost into an "easy" component (the rentals) and a "hard" component (the loads). We require the algorithm to be constant-competitive with respect to the easy component and allow an $O(\log k)$ ratio only with respect to the hard component.

Because of the working set phenomenon, researchers have tried designing cache systems that in a certain sense add such a renting ability. One practical implementation is to reserve the main cache for the supposed working set while adding a second, smaller cache of potential working-set candidates [JS97].

The k-server problem The question of the best possible competitive ratio for the k-Server problem of Manasse, McGeoch, and Sleator [MMS90] remains a major open question.

Problem k-Server The algorithm is given a metric space and an initial selection of k points where it has **servers**. It faces a sequence of requests to points in the space. When it receives a request, the algorithm must choose a server to move to the requested point. The goal is to minimize the total distance traveled by the servers.

Notice that a k-Server instance on a space of $k + 1$ points is easily modeled as an MTS problem instance with $k + 1$ points. In particular, each of the $k + 1$ points corresponds to a page that is *not* in the cache — the cache holds all other pages but the state's corresponding page.

Koutsoupias and Papadimitriou's proof that the Work-Function algorithm achieves an $O(k)$ competitive ratio was a breakthrough result, especially given the $\Omega(k)$ lower bound for deterministic algorithms [MMS90, KP95]. It is conceivable, however, that a randomized algorithm could achieve a $polylog(k)$ ratio. Hope that this might be possible comes from the $polylog(n)$ MTS result in Theorem 4.8. At the core of Theorem 4.8 is an algorithm for achieving an $O(r + \log n)$ ratio for the r-unfair MTS problem. Our goal of $O(r + \log k)$ for r-unfair Paging can be thought of as an extension of the $O(r + \log n)$ r-unfair MTS bound. This could potentially be one step toward achieving a $polylog(k)$ bound for k-Server. (Of course, there are many additional issues involved in attempting to construct such a recursive k-Server algorithm.)

7.2 A universe of $k + 1$ pages

Before we look at the general case where there can be arbitrarily many pages requested, we first restrict our attention to the simpler case where the request sequence can only include one more page than can be held in the cache (although any of these pages can of course be requested arbitrarily many times, in any order). This restricted case illustrates some of the ideas that appear in our general result.

Because of the close relationship of the $(k + 1)$-point case and metrical task systems, our result here can be seen as being an alternative to the two good algorithms for the MTS problem we have already seen, Share and Odd-Exponent. This new algorithm is simpler to describe and to analyze than the others, though the constants are slightly worse. It is a combination of Marking and Hedge.

Algorithm Phased-Hedge Each phase proceeds until every one of the $k + 1$ pages has had r requests. At the beginning of the phase, we associate to each page a weight w_i, initialized to 1. The weights w_i define a probability distribution $p_i = w_i/W$, where $W = \sum_j w_j$; this is our probability over pages *not* to have in the cache. (For example, initially all weights are 1 and so each page is equally likely to be the one outside the cache.) When a page is requested, we multiply the page's weight by β (a parameter of the algorithm) and readjust our probability distribution accordingly. (This effectively increases the probability that the page is in the cache.)

In the terminology of the machine learning literature, we could think of having an "expert" associated to each of the $k + 1$ subsets of k pages advocating that the cache contain these k pages, and we could think of Phased-Hedge as Hedge with the small modification that we reinitialize the algorithm periodically at phase boundaries.

Theorem 3.2 states that the expected loss incurred by Hedge is at most

$$\frac{\ln 1/\beta}{1-\beta} L + \frac{1}{1-\beta} \ln n \ ,$$

where L is the loss of the best expert in hindsight and n is the number of experts. In our context, this implies that the expected cost of the Phased-Hedge algorithm per phase is at most $1 + (r \ln(1/\beta) + \ln(k+1))/(1-$

$\beta)$. (The "1+" is the initialization cost for choosing a random page at the phase's beginning.) Now, noting that the off-line algorithm must pay at least 1 per phase, either to evict a page or to rent a page r times, we have the following theorem.

Theorem 7.2 *The competitive ratio of the* Phased-Hedge *algorithm for the r-unfair $(k+1)$-page* Paging *problem is at most*

$$\frac{\ln 1/\beta}{1-\beta} r + \frac{1}{1-\beta} \ln(k+1) + 1 \, .$$

For $\beta = \frac{3}{4}$, the bound of Theorem 7.2 is approximately $1.15r + 4\ln(k+1) + 1$. As β approaches 1, the bound approaches $\left(1 + \frac{\varepsilon}{2}\right) r + \frac{1}{\varepsilon} \ln k + 1$ for $\varepsilon = 1 - \beta$.

For Paging on more than $k+1$ pages, we extend the Phased-Hedge algorithm to have one "expert" for every *subset* of pages marked in the previous phase, which the expert predicts should be kept in the cache during the current phase. (A page is marked in a phase if it is requested at least r times, and a phase ends when k pages are marked.) Ignoring implementation issues, the two difficulties that this approach entails are first, that there are now many more experts, and second, that the possible cost for switching between two different experts increases from 1 to k. We deal with the first issue by giving a nonuniform initial weighting to the experts. The second issue involves substantially more effort.

7.3 The general case: Phases and the off-line cost

We begin our analysis of the general case by defining the notion of "phase" that the on-line algorithm uses and proving a lower bound for the off-line cost based on this notion. Then in Section 7.4 we describe how the algorithm behaves within each phase and prove an upper bound on the expected on-line cost. Because our on-line algorithm is not a "lazy" algorithm, we separately analyze its expected number of page faults (the easier part of the analysis) and its expected cost for modifying its probability distribution over caches (the harder analysis). To define the initial state of our problem, we assume the cache is empty before the first request occurs.

Like the Marking algorithm, we divide the request sequence into phases. We say that page j is *marked* when it has accumulated at least r requests within the phase. The phase reaches its end when any k pages become marked.

Let M^i denote the set of pages marked in phase i. (Define M^0 to be the empty set.) Also, let $\boldsymbol{\ell}^i_j$ denote the number of requests to page j in phase i. We define $\mathbf{m}^i$ as the number of pages marked in phase i but not in the previous phase ($|\mathrm{M}^i \setminus \mathrm{M}^{i-1}|$). Finally, we define $\mathbf{o}^i$ as the total off-line cost for renting pages outside $\mathrm{M}^{i-1} \cup \mathrm{M}^i$; that is, $\mathbf{o}^i = \frac{1}{r} \sum_{j \notin \mathrm{M}^{i-1} \cup \mathrm{M}^i} \boldsymbol{\ell}^i_j$.

As in the standard analysis of Marking, this use of phases gives a convenient lower bound on the off-line player's cost.

Lemma 7.3 *If $cost_{\mathrm{OPT}}(\boldsymbol{\sigma})$ is the optimal off-line cost for the task sequence, then we have*

$$cost_{\mathrm{OPT}}(\boldsymbol{\sigma}) \geq \frac{1}{2} \sum_i \left(\mathbf{m}^i + \mathbf{o}^i\right) \, .$$

Proof. Consider two phases $i-1$ and i together. Notice that for all but the k pages in the off-line cache at the beginning of phase $i-1$, the off-line algorithm must either load the page into its cache, at a cost of at least 1, or service all requests to that page (if any) by renting, at a cost of at least

$(\boldsymbol{\ell}_j^{i-1} + \boldsymbol{\ell}_j^i)/r$. Therefore, any off-line algorithm must pay at least

$$cost_{\mathrm{OPT}}(\boldsymbol{\sigma}^{i-1}\boldsymbol{\sigma}^i) \geq \left(\sum_j \min\left\{1, \frac{\boldsymbol{\ell}_j^{i-1} + \boldsymbol{\ell}_j^i}{r}\right\}\right) - k$$

in these two phases. For pages j marked in phases $i-1$ or i, we know $\boldsymbol{\ell}_j^{i-1} + \boldsymbol{\ell}_j^i \geq r$; for other pages j, we know $\boldsymbol{\ell}_j^i < r$ (and so $\boldsymbol{\ell}_j^i/r \leq 1$) since j is not marked in phase i. These facts imply

$$\begin{aligned}\left(\sum_j \min\left\{1, \frac{\boldsymbol{\ell}_j^{i-1} + \boldsymbol{\ell}_j^i}{r}\right\}\right) - k &\geq \left(\sum_{j \in \mathbf{M}^{i-1} \cup \mathbf{M}^i} 1\right) + \left(\sum_{j \notin \mathbf{M}^{i-1} \cup \mathbf{M}^i} \frac{\boldsymbol{\ell}_j^i}{r}\right) - k \\ &= (k + \mathbf{m}^i) + \mathbf{o}^i - k = \mathbf{m}^i + \mathbf{o}^i\ .\end{aligned}$$

Also note that any off-line player must pay at least $\mathbf{m}^1 + \mathbf{o}^1$ in the first phase. Let $\boldsymbol{\sigma}^i$ represent the sequence of requests in phase i. Then we get the following.

$$\begin{aligned}2cost_{\mathrm{OPT}}(\sigma) &\geq cost_{\mathrm{OPT}}\left((\boldsymbol{\sigma}^1\boldsymbol{\sigma}^2)(\boldsymbol{\sigma}^3\boldsymbol{\sigma}^4)\cdots\right) + cost_{\mathrm{OPT}}\left(\boldsymbol{\sigma}^1(\boldsymbol{\sigma}^2\boldsymbol{\sigma}^3)(\boldsymbol{\sigma}^4\boldsymbol{\sigma}^5)\cdots\right) \\ &\geq \left((\mathbf{m}^2 + \mathbf{o}^2) + (\mathbf{m}^4 + \mathbf{o}^4) + \cdots\right) + \left((\mathbf{m}^1 + \mathbf{o}^1) + (\mathbf{m}^3 + \mathbf{o}^3) + \cdots\right) \\ &= \sum_i (\mathbf{m}^i + \mathbf{o}^i)\ .\end{aligned}$$

■

7.4 The on-line algorithm

We now describe a randomized on-line algorithm whose expected cost in each phase i is $O(r + \log k)$ more than the off-line bound of $\frac{1}{2}(\mathbf{m}^i + \mathbf{o}^i)$ given in Lemma 7.3. To describe the algorithm, we use $\mathbf{p}_j^t$ to denote the probability that page j is in the cache after servicing the tth request. For ease of analysis, our algorithm may throw out (invalidate) pages in its cache even when there is no immediate need to do so, so $\sum_j \mathbf{p}_j^t$ may be less than k for some times t.

We divide the description and analysis of the algorithm into two parts. First, we describe how the algorithm determines the probabilities $\mathbf{p}_j^t$, and we use this to bound the expected number of page faults incurred by the algorithm. We then describe how the algorithm loads and ejects pages to maintain these probabilities, and we bound the additional cost incurred by those operations.

The on-line cache probabilities and expected number of page faults

The algorithm determines the probabilities $\mathbf{p}_j^t$ based on a weighted average over a collection of "experts". In phase i, we define an expert for each subset $A \subsetneq \mathbf{M}^{i-1}$ and give it an initial weight of $1/k^{k-|A|}$. The pages in the cache for this "expert" are the pages in the set A, plus up to the first $k - |A|$ pages not in $\mathbf{M}^{i-1}$ marked so far. Equivalently, we can think of the expert representing the following deterministic Paging algorithm:

- Initially, eject all pages in the set $\mathbf{M}^{i-1} \setminus A$ from the cache.
- On a page fault, rent the requested page if any of the following hold:

1. the page is in the set $\mathbf{M}^{i-1} \setminus A$,
2. the page has not yet become marked (it has received fewer than r requests in this phase),
3. the cache is full.

- Otherwise, on a page fault, load the requested page into the cache.

To determine the probabilities $\mathbf{p}_j^t$, we use the Hedge algorithm to update experts' weights, and we compute a weighted average of the experts' caches. Specifically, $\mathbf{p}_j^t$ is the result of dividing the total weight on experts having page j in their cache by the total weight on all the experts. We update the weights on the experts as in Hedge by penalizing them by a factor $\beta = \frac{1}{2}$ whenever they incur a page fault. If we select a cache according to a distribution matching these probabilities, then our algorithm's expected number of page faults will match the expected cost to Hedge.

One final addendum to the algorithm: If $\mathbf{m}^i = 0$ (i.e., the pages marked in this phase match the pages marked from the previous phase), then the off-line bound is $\mathbf{o}^i$ in this phase but some of the experts pay more than $r\mathbf{o}^i$ because they foolishly eject pages from their cache at the start for no reason. Therefore our algorithm also expects to pay more than $r\mathbf{o}^i$ and thus is not competitive. To handle this problem, our algorithm simulates the experts in a somewhat lazy manner. In particular, if an expert it is following says to eject a page but does not indicate a page to fill that slot, then the algorithm notes the recommendation but does not evict it until required. Nonetheless, we define the probabilities $\mathbf{p}_j^t$ as if we were immediately following the advice of the experts. The only case in which this turns out to be important is the case $\mathbf{m}^i = 0$.

Lemma 7.4 *By combining these experts using Hedge, the on-line algorithm's expected number of page faults in phase i is at most* $(\mathbf{m}^i + \mathbf{o}^i)(2.8r + 2\ln k + 1.1)$.

Proof. The case $\mathbf{m}^i = 0$ (when $\mathbf{M}^i = \mathbf{M}^{i-1}$) is a special case so we handle it first. In this case we use the fact that our algorithm is lazily following the experts' advice and that for $\mathbf{m}^i = 0$, no expert will recommend loading any pages into the cache. Therefore, the algorithm will have $\mathbf{M}^{i-1} = \mathbf{M}^i$ in its cache throughout the phase, paying a total of $r\mathbf{o}^i$, meeting the desired bound. In the following, then, we assume $\mathbf{m}^i > 0$.

One of the experts will do quite well, in particular the expert with $A = \mathbf{M}^{i-1} \cap \mathbf{M}^i$. This expert "knows" which of the marked pages from the previous phase should remain for the current phase, and it will not eject these. Note that this expert's initial weight is $1/k^{\mathbf{m}^i}$.

This good expert makes at most $2r\mathbf{m}^i + r\mathbf{o}^i$ page faults in the phase: For each of the $k - \mathbf{m}^i$ pages $j \in \mathbf{M}^i \cap \mathbf{M}^{i-1}$, it incurs 0 page faults because $j \in A$. For each of the $\mathbf{m}^i$ pages $j \in \mathbf{M}^i \setminus \mathbf{M}^{i-1}$, it incurs a total of r page faults until the page is finally marked and brought into the cache. For each of the $\mathbf{m}^i$ pages $j \in \mathbf{M}^{i-1} \setminus \mathbf{M}^i$, the renting cost is $\boldsymbol{\ell}_j^i$, which we know is less than r since j is not marked in phase i. Finally, the expert always rents pages $j \notin \mathbf{M}^{i-1} \cup \mathbf{M}^i$, and the total renting cost for these is $r\mathbf{o}^i$.

Theorem 3.2 for the loss of the Hedge algorithm can be generalized to the case of experts with unequal initial weights. In this case, the bound becomes

$$\frac{\ln 1/\beta}{1-\beta} L + \frac{1}{1-\beta} \ln \frac{W}{w} , \tag{7.1}$$

where w is the initial weight of the best expert in hindsight (and, as before, L is the loss of that expert) and W is the sum of the initial weights. In our case, if we choose $\beta = \frac{1}{2}$ and maintain probabilities $\mathbf{p}_j^t$ according to the expert weights as above, then the total expected number of page faults is at most

$$1.4(2r\mathbf{m}^i + r\mathbf{o}^i) + 2\ln \frac{W}{w_A} , \tag{7.2}$$

where W is the total of the experts' initial weights and $w_A = 1/k^{\mathbf{m}^i}$ is the weight for expert A. Since for each m between 1 and k, there are $\binom{k}{m}$ experts of weight k^{-m}, the total weight W is at most $\sum_{m=1}^{k} \frac{1}{m!} \leq e-1$. Thus (7.2) is at most

$$\begin{aligned} & 2.8r(\mathbf{m}^i + \mathbf{o}^i) + 2(\mathbf{m}^i \ln k + \ln(e-1)) \\ \leq & \quad (\mathbf{m}^i + \mathbf{o}^i)(2.8r + 2\ln k + 1.1) \, . \end{aligned}$$

■

One additional nonobvious fact about our use of the Hedge algorithm is the following.

Lemma 7.5 *If there is a request to page j at time t, then $\mathbf{p}_j^{t+1} \geq \mathbf{p}_j^t$ and for all $i \neq j$, $\mathbf{p}_i^{t+1} \leq \mathbf{p}_i^t$.*

Proof sketch. The easy part of the lemma is the statement that when a request is made to page j, the probability that page j is in the cache increases. That happens because Hedge penalizes all experts that do not have j in their cache and does not penalize those that do. The harder part is the statement about pages $i \neq j$; in particular, perhaps some pages are correlated.

Consider any fixed $m < k$. Let W_1 be the weight on experts for m-sets containing j , W_β be the weight on experts for m-sets not containing j, $W_{1,i}$ be the weight on experts for m-sets containing i and j, $W_{\beta,i}$ be the weight on experts for m-sets containing i but not j. We want to show that

$$\frac{W_{1,i} + W_{\beta,i}}{W_1 + W_\beta} \leq \frac{W_{1,i} + \beta W_{\beta,i}}{W_1 + \beta W_\beta} \, .$$

This follows if we can show $W_{1,i} W_\beta \leq W_{\beta,i} W_1$. Let M be the set of pages marked in the previous phase. Consider the instant before the request, and let o be the number of requests to pages outside M and $\ell_{i'}$ be the number of requests to each page $i' \in M$. Observe that the expert for a set A has accumulated loss $o + \sum_{i' \in M \setminus A} \ell_{i'}$ and so its weight w_A is $\beta^{o + \sum_{i' \in M \setminus A} \ell_{i'}}$. The proof uses this fact to show that each term on the left-hand side $W_{1,i} W_\beta$ corresponds to a term on the right-hand side $W_{\beta,i} W_1$. ■

Moving between probabilities

At any point in time, our algorithm maintains a probability distribution q over caches (experts), which induces page probabilities p_j over pages. The section above describes one distribution q using the Hedge algorithm. Notice, however, that for the purpose of computing the expected number of page faults (as in Lemma 7.4), any two distributions over caches that induce the same page probabilities are equivalent. Therefore, we are free to deviate from the instructions given by the Hedge algorithm so long as we are faithful to the page probabilities p_j. This is important for the next part of our analysis, where we bound the expected cost incurred by moving between probability distributions.

In particular, we now examine the following question. Given a current distribution q over caches that induces probabilities p_j over pages, and given a new target set of page probabilities p'_j that satisfies $\sum_j p'_j \leq k$, we want to move to some new distribution q' over caches that induces p'. At a minimum, any algorithm must load an expected $\sum_{p'_j > p_j} \left(p'_j - p_j\right)$ number of pages to move from the page probabilities p to p'. Achieving this is easily possible in a setting where $\sum_j p_j = 1$ (e.g., the case of $k+1$ pages total in which p_j represents the probability that page j is *not* in the cache) but it is harder in our setting, where $\sum_j p_j$ is as large as k. In this section, we show a method for achieving an expected cost of at most $2\sum_{p'_j > p_j} \left(p'_j - p_j\right)$.

A simple example will help illustrate the difficulty and the algorithm. Say that $k = 2$ and initially our cache is $[A, B]$ with probability $\frac{1}{2}$ and $[C, D]$ with probability $\frac{1}{2}$. This induces page probabilities p; say we want to convert this to a new distribution p' as follows.

page	A	B	C	D
p	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
p'	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$

If we momentarily forget about the cache capacity of k, we can easily move to a new cache distribution $\hat{q}$ consistent with p': we can simply evict B with probability $\frac{1}{2}$ if our cache is $[A, B]$ and load A with probability $\frac{1}{2}$ if our cache is $[C, D]$. So $\hat{q}$ is the following.

cache	$[A]$	$[A, B]$	$[C, D]$	$[A, C, D]$
$\hat{q}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

The $[A, C, D]$ possibility, unfortunately, exceeds the size limit of $k = 2$. However, there is (and there must be) a cache that has a vacancy, in this case $[A]$. We rebalance by adding page D to the small cache and evicting D from the large cache. This new cache distribution now includes only legal caches, and we use this for q'.

cache	$[A, D]$	$[A, B]$	$[C, D]$	$[A, C]$
q'	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

In other words, the strategy in this case is: "if our cache is $[A, B]$ then with probability $\frac{1}{2}$ do nothing and with probability $\frac{1}{2}$ evict B and load D; if our cache is $[C, D]$ then with probability $\frac{1}{2}$ do nothing and with probability $\frac{1}{2}$ evict D and load A." This strategy seems a bit strange because $p'(D) = p(D)$ yet we sometimes evict or load D, but this is necessary in this situation. As you can see, the expected number of page loads in this example is $\frac{1}{2}$, which equals $2 \sum_{p'_j > p_j} \left(p'_j - p_j\right)$.

Our strategy, in general, is as follows. To move from a set of probabilities p to p', for any page j with $p'_j < p_j$, we evict j from our cache (if present) with probability $1 - p'_j/p_j$. Next, for pages with $p'_j > p_j$, we add them to a cache not containing j with probability $(p'_j - p_j)/(1 - p_j)$. This gives us a cache distribution $\hat{q}$ with the correct probabilities p' and loading cost $\sum_{p'_j > p_j} (p'_j - p_j)$, but it may create caches that are too large.

Fortunately, the expected number of pages in the cache is $\sum p'_j \leq k$. Thus, if there are caches with more than k pages, there must be caches with fewer than k pages. Take a cache with more than k pages and one with fewer than k pages, and some page that is in the larger but not the smaller. We can evict the page from the larger cache and load it to the smaller cache in such a way as to not change p'. If the two caches do not have equal probabilities, we cannot immediately reduce the probability of both of the original caches to 0. However, one of the two caches will end with probability 0, and thus we are always making discrete progress in decreasing the total excess and shortage in cache sizes, over all caches with nonzero probability. Furthermore, the total probability of performing a load in the rebalancing step is no more than the probability of loading a page from in the increase step, since each load required for a rebalance originates from an increased probability. The expected number of loads is no more than $2 \sum_{p'_j > p_j} \left(p'_j - p_j\right)$.

Lemma 7.6 *Given a probability distribution q on caches, this implies page probabilities p. Given a new set of page probabilities p', we can move to a new probability distribution q' on caches with expected cost* $2 \sum_{p'_j > p_j} \left(p'_j - p_j\right)$.

Bounding the on-line movement cost

The final step to showing that our algorithm achieves the required bound is to bound what the algorithm pays to load pages in maintaining the page probabilities $\mathbf{p}^t_j$. We do this by employing Lemma 7.6 to bound this cost in terms of the expected number of page faults analyzed in Section 7.4.

Lemma 7.7 *Using the movement strategy given in Lemma 7.6, the expected loading cost for the probability sequence used in Lemma 7.4 is at most* $(\mathbf{m}^i + \mathbf{o}^i)(2.8r + 2\ln k + 1.1)$.

Proof. Consider the expert weights before receiving a request to page j. Let p be the page probabilities before the request and p' be the page probabilities after the request. Since j is the only page whose probability of being in the cache increases (Lemma 7.5), the expected loading cost from Lemma 7.6 is at most $2\left(p'_j - p_j\right)$.

We want to bound $p'_j - p_j$. Let x be the total weight on experts who have probability 1 on j and let y be the total weight on experts who have probability 0 on j. Since each expert in the first set has a loss of 0, the request will not alter their weights. Experts in the second set, however, experience a loss of 1, so their total weight decreases to $\beta y = y/2$.

$$\begin{aligned} p'_j - p_j &= \frac{x}{x + y/2} - \frac{x}{x+y} \\ &= \frac{xy/2}{(x+y)(x+y/2)} \\ &\leq \frac{1}{2}\frac{y}{x+y} = \frac{1}{2}(1 - p_j) \end{aligned}$$

This $1 - p_j$ is exactly the probability of faulting on the request. Thus our expected loading cost (at most $2(p'_j - p_j)$) is at most the expected number of page faults. The lemma follows from the bound of Lemma 7.4. ∎

Bounding the total expected on-line cost using Lemmas 7.4 (renting cost) and 7.7 (loading cost), and bounding the off-line cost using Lemma 7.3, we conclude with our competitive ratio of $O(r + \log k)$.

Theorem 7.8 *There is an algorithm whose r-unfair competitive ratio for Paging is* $8(2.8r + 2\ln k + 1.1)$.

Chapter 8

Conclusion

The metrical task system problem is one of the fundamental on-line problems in computer science. In this thesis, we have seen how its applications include machine learning and process migration. The thesis has neglected to mention its theoretical applications to other on-line problems like robot navigation and file migration.

We have seen how one can achieve much-improved asymptotic guarantees for metrical task systems. While the general-metric result is not immediately useful for actual systems, along the way we learned about algorithms for the uniform metric that do have practical promise, like Share and Odd-Exponent. The process migration experiment (Section 6.3) bolsters the feeling that these can be useful alternatives to Marking.

8.1 Themes

On our way to achieving improved results, we have seen three themes develop that may apply to more on-line analysis. The first is the useful relationship between a fundamental machine learning theory problem, Experts, and competitive analysis, especially with the unfair MTS problem. The Experts results have much promise as important tools to solving on-line problems; we have seen how it touches on MTS, Combine-Online, and Paging, but it is likely to have uses elsewhere. The Experts problem deserves to be included with MTS and k-Server as foundations for on-line analysis of algorithms.

Another theme of this thesis is the use of unfairness to refine our on-line goals. Essentially, unfairness gives us the opportunity to prioritize different types of costs by introducing a trade-off parameter. We have seen applications to MTS, Experts, and Paging; in all cases, the tradeoff has been between moving between selections and sticking with the current selection. Whether the unfairness concept can be applied naturally to other problems remains to be seen.

Finally, we have seen the importance of metric space approximation in competitive analysis. The $polylog(n)$ metrical task system result is a significant, sophisticated illustration of the usefulness of HST approximation to competitive analysis and approximation algorithms. Besides being historically one of the first major results using Bartal's HST approximation, metrical task systems are also likely to endure as an instance where HST approximation allow us to do much better than we can without it.

8.2 Open questions

A number of open questions, touched on in the progress of this thesis, remain open.

Question 8.1 Can Bartal's $O(h \log n \log\log n)$ approximation factor of an arbitrary metric space by h-HSTs be improved to $O(h \log n)$ [Bar98]?

Question 8.2 Is there a metric space where one can achieve an $o(\log n)$ competitive ratio for MTS? Blum *et al.* prove that for any algorithm on any particular space, the competitive ratio is at least $\Omega(\sqrt{\log n / \log\log n})$ [BKRS92].

Question 8.3 Can we improve on the competitive ratio for the MTS problem on general metric spaces? This thesis proves $O(\log^5 n \log\log n)$ (Theorem 4.8); building on this result, Fiat and Mendel improve it to $O(\log^2 n \log^2 \log n)$ [FM00]. Both use the only known tractable approach to achieving sublinear bounds: building an algorithm for an HST. This approach has the shortcoming that the metric space approximation factor will not improve beyond $O(\log n)$, and the competitive ratio for the HST will not improve beyond $O(\log n)$, giving an inherent limit of $O(\log^2 n)$.

Question 8.4 We have seen a number of algorithms for the r-unfair MTS problem on a uniform metric, the best bound being $r + 2e \ln n$ achieved by Odd-Exponent. Can one get an $r + \ln n$ algorithm for this problem? And is there an intuitive explanation for why Odd-Exponent, with its peculiar structure, does so well?

Question 8.5 Example 5.2 shows that one can get arbitrarily close to a static adversary's performance for both List-Update and Dynamic-Tree, but the algorithms to do this are massively inefficient. Are there efficient algorithms to do the same?

Question 8.6 For the Bandits problem with a switching cost, Corollary 5.4 shows an algorithm that is an additive $O(\sqrt[3]{dnT^2 \ln n})$ from the gain of the best bandit. Can this be improved to $O(\sqrt{dnT \ln n})$, as Auer *et al.* achieve for the problem with no switching cost [ACBFS98]?

Question 8.7 The paging algorithm of Chapter 7 requires exponential running time. Is there an efficient method achieving the same $O(r + \log n)$ guarantee?

Question 8.8 Can one achieve a guarantee of $r + O(\log n)$ for r-unfair Paging? Or perhaps $(1+\varepsilon)r + O(\frac{1}{\varepsilon}\log n)$? And can such an algorithm for the unfair scenario be used for k-Server on an HST space? We were able to abstract lower levels for MTS, but determining the proper way to do this for k-Server is a challenging problem. For instance, it appears that such an abstraction would have to to encourage multiple servers to be at a single point in the uniform space.

Question 8.9 For that matter, is there any way of using randomization to improve the $2k - 1$ ratio for k-Server achieved by Koutsoupias and Papadimitrou [KP95]? The conjecture is that $O(\log k)$ is possible, but we appear very far from any sublinear guarantee.

Bibliography

[ABM93] Y. Azar, A. Broder, and M. Manasse. On-line choice of on-line algorithms. In *Proc ACM-SIAM Symposium on Discrete Algorithms*, pages 432–440, January 1993.

[ACBFS95] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proc IEEE Symposium on Foundations of Computer Science*, pages 322–331, 1995.

[ACBFS98] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. *Submitted for publication*, 1998. Based on [ACBFS95].

[ACN96] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. In *Proc 4th European Symposium on Algorithms*, pages 419–430. Springer-Verlag, 1996.

[AvSW95] S. Albers, B. von Stengel, and R. Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

[Bar96] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proc IEEE Symposium on Foundations of Computer Science*, pages 183–193, October 1996.

[Bar98] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proc ACM Symposium on Theory of Computing*, pages 161–168, May 1998.

[BB97] A. Blum and C. Burch. On-line learning and the metrical task system problem. In *Proc ACM Workshop on Computational Learning Theory*, pages 45–53, 1997.

[BBBT97] Y. Bartal, A. Blum, C. Burch, and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proc ACM Symposium on Theory of Computing*, pages 711–719, 1997.

[BBF+90] A. Blum, A. Borodin, D. Foster, H. Karloff, Y. Mansour, P. Raghavan, M. Saks, and B. Schieber. Randomized on-line algorithms for graph closures. Personal communication, 1990.

[BBK99] A. Blum, C. Burch, and A. Kalai. Finely-competitive paging. In *Proc IEEE Symposium on Foundations of Computer Science*, pages 450–457, 1999.

[BDBK+94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BEY98] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University, 1998.

[BKRS92] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and lower bounds for randomized server problems. In *Proc IEEE Symposium on Foundations of Computer Science*, pages 197–207, 1992.

[BLS92] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *J of the ACM*, 39(4):745–763, 1992.

[BM85] J. Bentley and C. McGeoch. Amortized analysis of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[BRS97] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J Computing*, 26(1):110–137, 1997.

[Esk90] M. Eskicioglu. Process migration in distributed systems: A comparative survey. Technical Report TR 90-3, University of Alberta, January 1990.

[FKL+91] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *J of Algorithms*, 12:685–699, 1991.

[FM00] A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. In *Proc ACM Symposium on Theory of Computing*, 2000. To appear.

[FS97] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J Comp Syst Sci*, 55(1):119–139, 1997.

[HLS96] D. Helmbold, D. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proc ACM/IEEE International Conference on Mobile Computing and Networking*, 1996.

[HW98] M. Herbster and M. Warmuth. Tracking the best expert. *Machine Learning*, 32(2), August 1998.

[Ira91] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, June 1991.

[IS98] S. Irani and S. Seiden. Randomized algorithms for metrical task systems. *Theoretical Computer Science*, 194(1–2):163–182, March 1998.

[JS97] L. John and A. Subramanian. Design and performance evaluation of a cache assist to implement selective caching. In *Proc International Conference on Computer Design*, pages 610–518, October 1997.

[Kar90] R. Karp. A $2k$-competitive algorithm for the circle. Manuscript, August 1990.

[KMMO90] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proc ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[KP95] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *J of the ACM*, 42(5):971–983, September 1995.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LW94] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[Mit82] T. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[MMS90] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for server problems. *J Algorithms*, 11:208–230, 1990.

[Sei99] S. Seiden. Unfair problems and randomized algorithms for metrical task systems. *Information and Computation*, 2:219–240, February 1999.

[ST85a] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, February 1985.

[ST85b] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J of the ACM*, 32:652–686, 1985.

[Tei93] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47:5–9, 1993.

[Tom97] A. Tomkins. *Practical and Theoretical Issues in Prefetching and Caching*. PhD thesis, Carnegie Mellon University, October 1997. CMU-CS-97-181.

[Vov95] V. Vovk. A game of prediction with expert advice. In *Proc ACM Workshop on Computational Learning Theory*, pages 371–383, 1995.

Index