, , 1–24 ()

On-line Learning and the Metrical Task System Problem *

AVRIM BLUM AND CARL BURCH {avrim+,cburch+}@cs.cmu.edu
Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213

Editor:

Abstract. The problem of combining expert advice, studied extensively in the Computational Learning Theory literature, and the Metrical Task System (MTS) problem, studied extensively in the area of On-line Algorithms, contain a number of interesting similarities. In this paper we explore the relationship between these problems and show how algorithms designed for each can be used to achieve good bounds and new approaches for solving the other. Specific contributions of this paper include:

- An analysis of how two recent algorithms for the MTS problem can be applied to the problem of tracking the best expert in the "decision-theoretic" setting, providing good bounds and an approach of a much different flavor from the well-known multiplicative-update algorithms.
- An analysis showing how the standard randomized Weighted Majority (or Hedge) algorithm can be used for the problem of "combining on-line algorithms on-line", giving much stronger guarantees than the results of Azar *et al* [1] when the algorithms being combined occupy a state space of bounded diameter.
- A generalization of the above, showing how (a simplified version of) Herbster and Warmuth's weight-sharing algorithm can be applied to give a "finely competitive" bound for the uniform-space Metrical Task System problem. We also give a new, simpler algorithm for tracking experts, which unfortunately does not carry over to the MTS problem.

Finally, we present an experimental comparison of how these algorithms perform on a process migration problem, a problem that combines aspects of both the experts-tracking and MTS formalisms.

Keywords: On-line learning, Metrical task systems, Combining expert advice, Randomized on-line algorithms

1. Introduction

The problem of combining expert advice has been studied extensively in the Computational Learning Theory literature (e.g., [6, 10, 13]). In this problem, an on-line learning algorithm has access to n advisors (the "experts") and must decide which of their advice to follow in making a series of decisions. A problem with similar flavor is the Metrical Task System problem studied in the area of On-line Algorithms. In this problem an on-line algorithm is in one of n "states", and must decide how to move among these states to best process a series of tasks being received. (We describe these two problems more fully below.) In this paper we examine relations between these problems and use these relations to construct new algorithms for

* Avrim Blum is supported in part by NSF National Young Investigator grant CCR-9357793. Carl Burch is supported in part by a National Science Foundation Graduate Fellowship.

each and derive new guarantees. To help structure the discussion and to frame some of the issues that arise, we begin with two motivating scenarios.

Scenario 1 (process migration). Consider a process running on some machine in a network that has the ability to *migrate*: if it finds that its machine is heavily loaded compared to another machine in the network, it may decide to move to the other machine in order to work faster. Say that each minute the process gets load-average information from all n machines in its network (e.g., by pinging them), and it may use this information to decide what to do next. What is a good strategy for deciding when and where this process should move?

Scenario 2 (combining on-line algorithms on-line [1]). Consider an on-line problem, such as paging, where we have several plausible on-line algorithms to choose from (maybe FIFO, LRU, and LFU). Suppose we do not know *a priori* which of these will perform best on an upcoming sequence of requests. Can we combine these algorithms in some generic way into an on-line strategy whose performance will never be too much worse than the best of them in hindsight?

Scenario 1 could be modeled in the "decision-theoretic" experts framework of Freund and Schapire [10] and Chung [7] as follows. We view the machines as *experts* and the loads on the machine as *losses*. An unloaded machine has loss 0, indicating that the process would have wasted no time if it had been on that machine, and a heavily loaded machine (not counting the load caused by our process itself) has loss approaching 1, indicating the process would have wasted nearly the entire minute had it been on that machine. At each time step t, based on information about the past, the process probabilistically selects a machine to use using some distribution $\mathbf{p}^t$ and then experiences an expected loss $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$, where $\boldsymbol{\ell}^t$ is the vector of losses. Over time, the total expected loss (time wasted) is $\sum_t \mathbf{p}^t \cdot \boldsymbol{\ell}^t$, which we would like to minimize. Unfortunately, moving between machines is not free: it takes time to transfer one's state. Thus, a better model for our scenario would be a version of the experts setting in which not only does an algorithm incur loss by selecting a bad expert, but it also incurs loss for moving between experts.

Scenario 2 can also naturally be modeled by viewing each on-line algorithm as an "expert" dispensing advice at each time step about what to do next. In fact, this scenario is simpler than the first one in the sense that here we explicitly compete against (compare ourselves to) the best single expert, whereas in Scenario 1 we might have the higher hope of doing nearly as well as the best strategy that is allowed to switch several times. On the other hand, in Scenario 2, each expert has some internal state (e.g., the contents of its cache) that affects how it is charged on each request. Thus if we want to switch from following the advice of expert i to following the advice of expert j *and also* to match j's current internal state, we may also need to incur some kind of "movement cost" (we will discuss this more fully in Section 3.3).

This notion of an on-line algorithm having *state*, with a cost for moving between states, is captured by the *Metrical Task System* (MTS) problem studied in the On-line Algorithms literature [5, 12]. In this problem, we imagine the on-line algorithm as controlling a system that can be in one of n *states* or *configurations*, with some distance metric d among the states specifying the cost of moving from one state to another. This system incrementally receives a sequence of *tasks*, where each task has a *cost vector* specifying the cost of performing the task in each state of the system. Given a new task, an algorithm must decide whether to process the task in the current state (paying the amount specified in the cost vector) or to move to a new state and process the task there (paying both the movement cost and the amount specified in the cost vector for the new state).

The MTS problem is like a version of the decision-theoretic experts problem where there is a cost to switch between experts, but where there is also one-step lookahead (the algorithm sees a task before deciding how to process it, instead of choosing an expert before seeing their losses). One feature of the MTS setting is that the movement cost allows comparison of an on-line algorithm to the best off-line strategy that is allowed to move between states *arbitrarily*, a more ambitious goal than comparing to the best single state. That is, one can compare the performance of an on-line algorithm to the best possible cost given the entire sequence of cost vectors in advance. This measure, and in particular the worst-case ratio over task sequences of the algorithm's performance to the optimal off-line performance, is called the *competitive ratio* of the algorithm.

This paper begins in Section 2 by more precisely defining the problems and goals of the experts and MTS settings. In Section 3 we describe a general relationship between the settings, and discuss the problem of combining on-line algorithms as a "warm-up" (in particular as a warm-up to Section 5). In Section 4 we examine several algorithms designed for the MTS problem and show how these can be applied to the problem of tracking the best expert in the decision-theoretic setting. Section 5 explores the other direction: we analyze two algorithms for tracking the best expert in the decision-theoretic setting based on established weighted-experts algorithms [11, 13] and show that one of them achieves good performance for the MTS problem, whereas the other has an unbounded competitive ratio. Finally, in Section 6 we give an empirical comparison of these algorithms and others for the process migration scenario discussed above.

2. Definitions

2.1. *Tracking experts in the decision-theoretic setting*

The first setting we consider is the "decision-theoretic" framework for learning from expert advice (also called the on-line allocation problem) [10, 7]. In this problem the learning algorithm has access to n *experts* and faces a sequence of *trials*. For trial t, the algorithm chooses a probability distribution $\mathbf{p}^t$ over the experts. After choosing this distribution, a *loss vector* $\boldsymbol{\ell}^t \in [0,1]^n$ is revealed, and the learning algorithm is penalized with loss $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$. This penalty may be understood as the expected loss of the algorithm when it selects one expert with distribution $\mathbf{p}^t$ and then incurs that expert's loss. We call this the Experts-DTF setting.

This problem is typically analyzed with the goal of performing nearly as well as the best expert on the given sequence of trials. A stronger goal is the *partitioning bound* (considered by Herbster and Warmuth [11] and Littlestone and Warmuth [13] for specific classes of loss functions) of performing nearly as well as the best *sequence* of experts. Specifically, given a partition of the trial sequence into k segments, we define the *loss of the partition* as $\sum_{i=1}^{k} L[i]$, where $L[i]$ is the loss of the best expert in segment i. Our goal is to achieve a loss of at most

$$\min_P [a_1 L_P + a_2 k_P], \tag{1}$$

where L_P is the loss of partition P and k_P is the number of segments in P. We will seek to achieve this for a_1 and a_2 as small as possible. (The easier goal of competing against the best single expert is a restriction of the partitioning bound to the case $k_P = 1$.)

2.2. *The MTS problem and the competitive ratio*

In the *Metrical Task System (MTS) problem* [5, 12], an on-line algorithm controls a system with n *states* located at points in a space with distance metric d. The algorithm receives, one at a time, a sequence of *tasks*, each of which has a *cost vector* specifying the cost of performing the task in each state. Say the system currently occupies state i and, given the task vector $\boldsymbol{\ell}$, the algorithm tells the system to move to state j. Then the algorithm pays both the cost of moving to state j and the amount specified by the cost vector for processing the task in j; that is, the cost is $d_{i,j} + \ell_j$. (If $i = j$ then, since $d_{i,i} = 0$, the cost is just ℓ_j.)

In this paper we consider the simplest case of the MTS problem, in which we have a *uniform* metric space. In other words, $d_{i,j} = 1$ for all $i \neq j$ (and $d_{i,i} = 0$).

The MTS problem is typically analyzed by comparing the performance of an on-line algorithm to the best that one could have done in hindsight if one had known the task sequence in advance (called the *optimal off-line cost*). Specifically, say $cost_A(\sigma)$ is the cost to algorithm A for task sequence σ; if A is randomized, then for a fixed σ, $cost_A(\sigma)$ is a random variable. We say that algorithm A has *competitive ratio* a if, for some b, for all task sequences σ,

$$\mathbf{E}[cost_A(\sigma)] \;\leq\; a \cdot cost_{\mathrm{OPT}}(\sigma) + b\,, \tag{2}$$

where OPT is the optimal off-line algorithm (the optimal strategy in hindsight).

This definition is called the *oblivious adversary* model since the order of quantifiers $(\forall\sigma \mathbf{E}[cost_A(\sigma)])$ can be viewed as obligating an adversary to choose σ in advance, before seeing the outcome of A's coin flips. A consequence of this model is that the "state" of the randomized algorithm at a time t can be thought of as a probability distribution $\mathbf{p}^t$ over the states of the task system. Furthermore, to move from $\mathbf{p}^t$ to distribution $\mathbf{p}^{t+1}$ in the uniform metric space, the algorithm need pay only an (expected) cost of

$$d(\mathbf{p}^t, \mathbf{p}^{t+1}) \;=\; \sum_i \max\{0, p_i^t - p_i^{t+1}\}\,.$$

In other words, say at time t the randomized on-line algorithm has a probability distribution $\mathbf{p}^t$, receives a cost vector $\boldsymbol{\ell}^t$, and modifies its distribution to $\mathbf{p}^{t+1}$. Then its total (expected) cost is

$$d(\mathbf{p}^t, \mathbf{p}^{t+1}) + \mathbf{p}^{t+1} \cdot \boldsymbol{\ell}^t .$$

For a more "fine-grained" analysis of the MTS problem, we could split the optimal cost $cost_{\mathrm{OPT}}(\sigma)$ into the amount spent on processing tasks $localcost_{\mathrm{OPT}}(\sigma)$ and the amount spent for movement $movecost_{\mathrm{OPT}}(\sigma)$. We could then try for a bound of the form

$$\mathbf{E}[cost_A(\sigma)] \ \leq \ a_1 localcost_{\mathrm{OPT}}(\sigma) + a_2 movecost_{\mathrm{OPT}}(\sigma) + b, \tag{3}$$

optimizing a_1 and a_2 together. In fact, while for the uniform space the best possible bound of the form in equation (2) has $a = \Omega(\log n)$, it is possible to achieve a bound of the form in equation (3) with $a_2 = O(\log n)$ but $a_1 = O(1)$.

Within the On-line Algorithms literature, this kind of refined analysis has been examined under the notion of the *r-unfair competitive ratio* considered in Blum *et al* [4] and formalized explicitly by Seiden [14]. In the r-unfair MTS problem, OPT pays r times more than the on-line algorithm does for movement. That is, to move from state i to state j and then process a task in state j, the off-line player pays $rd_{i,j} + \ell_j$. This parameter r is known to the on-line and off-line MTS algorithms. Optimizing the r-unfair ratio is equivalent to optimizing $\max\{a_1, a_2/r\}$ in equation (3) for the standard MTS problem.

The notion of an r-unfair competitive ratio was initially developed to formalize the problem of recursively combining MTS algorithms in hierarchical metric spaces.[1] In fact, an algorithm for the r-unfair ratio on the uniform metric space is at the heart of the best algorithm known for the MTS problem on *arbitrary* metric spaces [3] (together with an approximation of arbitrary spaces by hierarchical ones [2]).

2.3. High-level comparison

Notice that the MTS problem is much like the Experts-DTF problem, but with a few key differences.

- 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-DTF 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 "farther" into the future. That is, an adversary trying to defeat an MTS algorithm might

as well break apart a large task vector into several small task vectors since it does not increase the optimal off-line cost and only delays the information given to the on-line algorithm.

Notice that in both settings, deterministic on-line algorithms have very poor worst-case bounds. In particular, an adversary who knows the deterministic algorithm could at each time step ensure that the expert (state) that the on-line algorithm is currently using has a loss of 1, while the others have loss of 0. In this case, in both the MTS and Experts-DTF settings, the on-line algorithm must pay 1 unit per time step, but in hindsight, by the pigeonhole principle, there must exist an expert (state) with average loss only $1/n$ per time step, for a ratio of n.

3. General relations and combining on-line algorithms

3.1. General relations

As alluded to above, the problem of achieving a good r-unfair competitive ratio for the uniform MTS problem is closely related to the partitioning bound in the Experts-DTF setting. Roughly, the "r" parameter allows one to trade off dependence on the loss L of the partition with dependence on the number of segments k the partition has. Formally, we have the following theorem.

THEOREM 1 *Let A be a randomized algorithm for the uniform MTS problem that, given r, achieves an r-unfair competitive ratio of $a_{n,r}$. Then in the Experts-DTF setting, given r, A will achieve total loss at most*

$$a_{n,r}(L + rk) + b\ ,$$

where L is the loss of the best k-segment partition, and b is a constant that may depend on r and n (typically, $b \leq r$).

Proof: Consider some sequence of trials σ. Let L_A denote the loss of algorithm A on σ and let c_A denote the expected cost of A on σ viewed as tasks in a uniform MTS problem (i.e., c_A is what A "believes" its cost is). In other words, if A is distributed according to $\mathbf{p}$, receives loss vector $\boldsymbol{\ell}$, and moves to distribution $\mathbf{p}'$, then A "believes" it is paying $d(\mathbf{p},\mathbf{p}') + \mathbf{p}' \cdot \boldsymbol{\ell}$ when in reality it is paying $\mathbf{p} \cdot \boldsymbol{\ell}$. Thus c_A and L_A may differ.

We begin by showing that $L_A \leq c_A$. Consider a single trial. Let $\ell_i \in [0,1]$ be the loss of expert i, and let p_i and p'_i represent the probability assigned to expert i before and after the trial. Then we can bound the loss on that trial as:

$$\begin{aligned} L_A &= \sum_i p_i \ell_i \\ &= \sum_i p'_i \ell_i + \sum_i (p_i - p'_i)\ell_i \\ &\leq \sum_i p'_i \ell_i + \sum_{i: p_i > p'_i} (p_i - p'_i)\ell_i \end{aligned}$$

$$\leq \sum_i p_i' \ell_i + \sum_{i:p_i > p_i'} (p_i - p_i')$$
$$= c_A \, .$$

Now consider a partition that switches between experts k times and incurs total loss L. The cost of the best off-line strategy for the partition is $L + rk$ in the r-unfair MTS setting. Thus we have $L_A \leq a_{n,r}(L + rk) + b$ (for some constant b) as desired. ■

In particular, an immediate consequence of Theorem 1 is that if $a_{n,r}$ is of the form $(1 + (\alpha \ln n)/r)$, then for any given $\epsilon > 0$, by setting $r = (\alpha \ln n)/\epsilon$, A can achieve total loss at most

$$(1+\epsilon)L + \left(\left(1 + \frac{1}{\epsilon}\right) \alpha \ln n\right) k + b$$

(where typically $b \leq \frac{\alpha \ln n}{\epsilon}$). We will examine such an algorithm, Odd-Exponent, in Section 4.4.

3.2. Competing against the best state

In the other direction, algorithms that achieve good bounds in the Experts-DTF setting may or may not perform well on the MTS problem, depending on how well they control their movement costs. We will discuss this more fully in Section 5. In this section we consider an easier "warm-up problem": showing that a specific algorithm (the WMR [13] or Hedge [10] algorithm) that competes well against the best *single* expert in the Experts-DTF setting, also achieves good bounds for the problem of competing against the "best single state in hindsight" in the MTS setting.[2] The essential issue is to show that the algorithm's movement cost is not too high. We then show how this can be applied to the problem of combining on-line algorithms on-line mentioned in Section 1.

We begin with a description of the algorithm.

Algorithm WMR *(Hedge).* The algorithm has one parameter $\beta \in [0, 1]$, and maintains a weight w_i for each expert, initialized to 1. At each time step the algorithm chooses an expert according to distribution $p_i = w_i / \sum_{j=1}^n w_j$. After receiving the loss vector $\boldsymbol{\ell}$, the algorithm multiplies each weight w_i by β^{ℓ_i}.

The following theorem bounds the loss of this algorithm in the Experts-DTF setting as function of the loss of the best expert in hindsight.

THEOREM 2 ([13, 10]) *If the best expert has total loss L, then* WMR *incurs loss at most*

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

We now analyze the algorithm's movement cost, showing that it is at most $\ln(1/\beta)$ times larger than its loss in the Experts-DTF setting.

THEOREM 3 *The* WMR *algorithm has the property that if it has probability distribution* $\mathbf{p}$*, receives loss vector* $\boldsymbol{\ell}$*, and then updates to probability distribution* $\mathbf{p}'$*, then*

$$d(\mathbf{p}, \mathbf{p}') \;\leq\; \ln(1/\beta)\mathbf{p} \cdot \boldsymbol{\ell}\,.$$

Proof: Let w_i denote the weight of expert i before the trial and w'_i denote the weight after. Let $W = \sum_i w_i$ and $W' = \sum_i w'_i$, so $p_i = w_i/W$ and $p'_i = w'_i/W'$. Then we have

$$\begin{aligned}
d(\mathbf{p}, \mathbf{p}') &= \sum_{i:p_i > p'_i} \left(\frac{w_i}{W} - \frac{w'_i}{W'} \right) \\
&\leq \sum_{i:p_i > p'_i} \left(\frac{w_i}{W} - \frac{w'_i}{W} \right) \\
&\leq \sum_i \left(\frac{w_i}{W} - \frac{w'_i}{W} \right) \\
&= \sum_i \frac{w_i}{W} (1 - \beta^{\ell_i}) \\
&\leq \sum_i p_i \left(\ell_i \ln(1/\beta) \right) \;,
\end{aligned}$$

which is our desired result. ■

COROLLARY 1 *In the MTS setting, if the best state has total cost* L*, then* WMR *incurs an expected cost at most*

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

which is at most

$$(1 + 2\epsilon)L + \left(\frac{7}{6} + \frac{1}{\epsilon} \right) \ln n$$

for $\epsilon = 1 - \beta \leq 1/4$*.*

Proof: The corollary follows by combining Theorems 2 and 3. The only technical issue involved here is that the MTS setting includes one-step lookahead (the algorithm pays $\mathbf{p}' \cdot \boldsymbol{\ell}$) and unbounded cost vectors, instead of zero-lookahead (the algorithm pays $\mathbf{p} \cdot \boldsymbol{\ell}$) and bounded cost vectors ($\ell_i \in [0, 1]$). The lookahead only helps, however, since the WMR algorithm has the property that for any given $\boldsymbol{\ell}$, $\mathbf{p}' \cdot \boldsymbol{\ell} \leq \mathbf{p} \cdot \boldsymbol{\ell}$. Thus if the largest component of $\boldsymbol{\ell}$ is bounded by 1, we get the desired bound. If k is a bound on all components of $\boldsymbol{\ell}$, then we can divide $\boldsymbol{\ell}$ into k

vectors of the form $\frac{1}{k}\boldsymbol{\ell}$; WMR's cost on these vectors obeys the desired bound and the final probability distribution is the same. By the triangle inequality on d, the cost incurred by WMR (with one-step lookahead) on $\boldsymbol{\ell}$ directly is at most its cost on the k pieces.

The conversion to terms of ϵ uses the inequality $-\ln(1-\epsilon) \leq \epsilon + 0.61\epsilon^2$ for $\epsilon \leq 1/4$. ■

3.3. Combining on-line algorithms on-line

The above analysis can be applied to the problem of combining on-line algorithms on-line, studied by Azar *et al* [1]. In this scenario, we have a collection of on-line algorithms $A_1, A_2, \ldots, A_n$ for some problem, such as paging, that can be viewed as a metrical task system. In other words, we assume that at each time step a "request" is received, then each algorithm A_i has the opportunity to change its internal state (at a cost), then each algorithm pays a cost which is a function of both the request and its (new) internal state. The "master" algorithm is allowed to simulate the A_i (thus, at any point in time it knows the internal state and total loss so far of each) and its goal is to perform nearly as well as the best of them on the request sequence.

For example, for the paging problem, the "state" of an algorithm is the contents of its cache and a "request" is a memory access. If the access is to an item residing in the cache (a *hit*) then the algorithm pays 0. If the access is to an item not in the cache (a *miss*), then the algorithm must throw out some page in its cache and replace it with the page containing the requested item, for a cost of 1.

Azar *et al* [1] present a method for combining n such on-line algorithms that is guaranteed on any request sequence to perform at most $O(\log n)$ times worse than the best of them. (See [1] for the precise statement of the bounds.) Using the WMR algorithm, we can do better, performing just $(1+\epsilon)$ times worse than the best of them (for any given $\epsilon > 0$), with an additive constant of $O(\frac{D}{\epsilon}\log n)$, under the assumption that the underlying metric space has bounded diameter D. Specifically, we assume there is an upper bound D on the cost for switching from one state (e.g., the state of algorithm A_i) to another (e.g., the state of algorithm A_j).[3] For instance, for the paging problem, the diameter of the space is the cache size: one can always move from the state of A_i to the state of A_j by dumping the contents of A_i's cache and loading in A_j's. In fact, in keeping with the MTS framework, when our global algorithm tells us to switch from A_i to A_j, then instead of doing something clever or "lazy", we will do exactly this, immediately switching to A_j's current internal state.

The bound for WMR follows immediately from Corollary 1.

THEOREM 4 *Given n on-line algorithms for a problem that can be formulated as a Metrical Task System of diameter at most $D > 0$, and given $\epsilon < 1/4$, the* WMR *algorithm can combine them such that on any request sequence σ it incurs expected cost at most*

$$(1+2\epsilon)L + \left(\frac{7}{6}+\frac{1}{\epsilon}\right) D \ln n ,$$

where L is the cost of the best of the n algorithms on request sequence σ.

Proof: Let us write our cost in units of D. That is, given cost vector $\boldsymbol{\ell}$, we will view it as $\tilde{\boldsymbol{\ell}} = \frac{1}{D}\boldsymbol{\ell}$. In these units, the space has diameter 1, so the bound of Corollary (1) applies. If we rewrite costs in terms of single units, the cost of WMR is multiplied by D, but so is the cost L of the best of the n algorithms. Thus, the factor of D appears only in the additive constant. ■

4. MTS-style algorithms

We now investigate several algorithms designed for the MTS problem and how they can be applied to the Experts-DTF setting.

To describe these algorithms, we need the notion of the *reduced loss* of an expert (usually called the *work function* in the On-line Algorithms literature). For a parameter r, the reduced loss $\hat{L}_i^t$ of expert i at time t is defined as follows. Initially $\hat{L}_i^0$ is 0. Given a loss vector $\boldsymbol{\ell}^t$ we update each expert's reduced loss to

$$\hat{L}_i^{t+1} = \min\{\hat{L}_i^t + \ell_i^t, \min_{j\neq i} \hat{L}_j^t + \ell_j^t + r\} .$$

(Using MTS terminology, $\hat{L}_i^t$ is the optimal off-line cost for servicing the first t tasks and ending at state i, if we are in the r-unfair setting where the off-line algorithm pays r to switch its state.) Notice that $\hat{L}_i$ is never greater than the actual total loss incurred by expert i, and furthermore $\hat{L}_i$ and $\hat{L}_j$ can never differ by more than r. The point of this definition is that by competing against the reduced losses (rather than against the potentially much higher true losses) we will be able to achieve a good partitioning bound.

If $\hat{L}_i = \hat{L}_j + r$, then so long as expert j does not incur a loss, expert i can incur losses with impunity without increasing its reduced loss. To emphasize this point, if one expert's reduced loss is r more than another's, we will say that the higher is *pinned* by the lower. Several MTS algorithms allocate probability to experts as a function of the reduced losses; such an algorithm must place zero probability on all pinned experts.

4.1. Two experts

The following algorithm Linear [4] achieves an optimal competitive ratio for the r-unfair MTS problem on two states. In the Experts-DTF setting, the algorithm can be viewed as follows.

Algorithm Linear *[4].* The algorithm has one parameter r. Let $\hat{L}_i$ represent the reduced loss of expert i with respect to r. The algorithm allocates

$$p_0 = \frac{1}{2} + \frac{\hat{L}_1 - \hat{L}_0}{2r}$$

probability to expert 0 and $p_1 = 1 - p_0$ probability to expert 1.

This strategy moves probability linearly between experts, so that an expert's probability is zero when it is pinned. This strategy's MTS performance problem turns out to be optimal and is summarized by the following theorem.

THEOREM 5 ([4]) Linear *has r-unfair competitive ratio $1 + 1/r$ on the two-state MTS problem.*

By combining this with Theorem 1, we get the corollary that the partitioning bound of Linear for two experts is at most

$$(1 + 1/r)L + (r + 1)k + b$$

(where in fact $b = r/2$). To achieve a somewhat better bound (and to illustrate the potential-analysis involved), we analyze the algorithm directly in the Experts-DTF setting.

THEOREM 6 *For integer parameter r, the loss of* Linear *for two experts is at most*

$$\left(1 + \frac{1}{2r}\right) L + \left(r + \frac{1}{2}\right) k$$

for any k-segment partition of loss L.

Proof: Consider segment i of the partition with a loss $L[i]$. Because the algorithm is symmetric, we may assume without loss of generality that the better expert for the segment is expert 0. (So $L[i]$ represents the total loss of expert 0 in the segment.) Let δ represent the fractional component of $\hat{L}_1 - \hat{L}_0$. We will use a potential function over this segment of[4]

$$\Phi = r p_1^2 + \frac{1}{2} p_1 + \frac{\delta(1-\delta)}{4r}.$$

Notice that Φ is always between 0 and $r + 1/2$. (If $\hat{L}_1 - \hat{L}_0 = -r + \delta$ for $\delta \in [0, 1]$, then $p_1 = 1 - \delta/2r$ and so $\Phi = r + 1/2 - \delta$.)

Say the algorithm receives loss vector $\langle \ell_0, \ell_1 \rangle$. Our goal is to show that the algorithm's cost plus potential change is at most $\ell_0(1 + \frac{1}{2r})$, because then the total cost for segment i would be at most $(1 + \frac{1}{2r})L[i]$ plus the maximum potential change between segments, $r + 1/2$. Thus the total cost for the partition would be at most $(1 + \frac{1}{2r})L + (r + \frac{1}{2})k$.

If $\ell_0 = \ell_1$, then the algorithm's cost plus potential change is just ℓ_0 and we are done. If not, then we may assume $\langle \ell_0, \ell_1 \rangle$ is 0 in one of its components because we can divide the vector into two pieces $\langle \hat{\ell}, \hat{\ell} \rangle$ and $\langle \ell_0 - \hat{\ell}, \ell_1 - \hat{\ell} \rangle$ for $\hat{\ell} = \min\{\ell_0, \ell_1\}$. This division does not affect the algorithm's cost or final probability distribution, and does not change $L[i]$. We split the remaining possibilities into four cases.

Case 1. The vector is $\langle \ell, 0 \rangle$ and $\hat{L}_1 - \hat{L}_0 \geq -r + \ell$. Then $\hat{L}_1 - \hat{L}_0$ decreases by ℓ and so p_0 decreases by $\ell/2r$ and p_1 increases by $\ell/2r$. Notice that the last term of the potential function increases by at most $\ell(1-\ell)/4r$. The amortized cost, then, is

$$p_0\ell + \Delta\Phi \le p_0\ell + \left(p_1\ell + \tfrac{\ell^2}{4r} + \tfrac{\ell}{4r} + \tfrac{\ell(1-\ell)}{4r}\right) = \ell\left(1 + \tfrac{1}{2r}\right) .$$

Case 2. The vector is $\langle \ell, 0\rangle$ and for some $\tilde{\ell} \in [0, \ell)$ we have $\hat{L}_1 - \hat{L}_0 = -r + \tilde{\ell}$. Then p_1 increases from $1 - \tilde{\ell}/2r$ to 1, and δ drops from $\tilde{\ell}$ to 0. The amortized cost is

$$p_0\ell + \Delta\Phi = \tfrac{\tilde{\ell}}{2r}\ell + \left(\tilde{\ell} - \tfrac{\tilde{\ell}^2}{4r} + \tfrac{\tilde{\ell}}{4r} - \tfrac{\tilde{\ell}(1-\tilde{\ell})}{4r}\right) \le \ell\left(1 + \tfrac{1}{2r}\right) .$$

Case 3. The vector is $\langle 0, \ell\rangle$ and $\hat{L}_1 - \hat{L}_0 \le r - \ell$. Then p_1 decreases by $\ell/2r$. The last term of the potential function increases by at most $\ell(1-\ell)/4r$. The amortized cost is

$$p_1\ell + \Delta\Phi \le p_1\ell + \left(-p_1\ell + \tfrac{\ell^2}{4r} - \tfrac{\ell}{4r} + \tfrac{\ell(1-\ell)}{4r}\right) = 0 .$$

Case 4. The vector is $\langle 0, \ell\rangle$ and for some $\tilde{\ell} \in [0, \ell)$ we have $\hat{L}_1 - \hat{L}_0 = r - \tilde{\ell}$. Then p_1 drops from $\tilde{\ell}/2r$ to 0, and, because r is integral, δ drops from $1 - \tilde{\ell}$ to 0. The amortized cost is

$$p_1\ell + \Delta\Phi = \tfrac{\tilde{\ell}}{2r}\ell + \left(-\tfrac{\tilde{\ell}^2}{4r} - \tfrac{\tilde{\ell}}{4r} - \tfrac{(1-\tilde{\ell})\tilde{\ell}}{4r}\right) \le 0 .$$

In all cases, the algorithm's cost is at most $\ell_0(1 + 1/2r)$. ∎

4.2. Work-Function

For the $(r = 1)$ MTS problem on more than two states, the Work-Function algorithm of Borodin, Linial, and Saks [5] is provably optimal for deterministic algorithms (even on general metrics).

Algorithm Work-Function *[5].* We maintain the reduced loss of each expert. When the current expert becomes pinned by another, the algorithm moves to the pinning expert. (In the uniform metric, this is the expert with least reduced loss.)

For example, say we have four experts whose current reduced losses are $\langle 5, 6, 5.9, 5\rangle$, and we are at the third expert. Given loss vector $\langle 0.5, 0.1, 0.6, 0.3\rangle$, Work-Function would update the reduced losses to $\langle 5.5, 6.1, 6.3, 5.3\rangle$. Since expert 3 is now pinned by expert 4, we would move to expert 4.

Because of the adversarial model of competitive analysis, deterministic algorithms have very poor competitive ratios. The best achievable guarantee for deterministic algorithms is $2n-1$ [5], and Borodin, Linial, and Saks show that Work-Function achieves this guarantee. This is much worse than the competitive ratios achieved with randomized algorithms, but in practice (including the experiments of Section 6) Work-Function can perform well.

4.3. Marking

A simple randomized algorithm for the uniform metric space is the Marking algorithm of Borodin, Linial, and Saks [5] and Fiat *et al* [9].

Algorithm Marking. We maintain a counter for each state. At the beginning of each phase, the counters are reset to 0, and the algorithm occupies a random state. Given a cost vector $\boldsymbol{\ell}$, we increment the ith counter by ℓ_i. When the counter for our current state reaches r, we move to a random state whose counter is less than r. If no such state exists, we begin a new phase by resetting the counters and going to a random state.

This algorithm was designed for the "fair" setting in which $r = 1$. For that case, its competitive ratio is $2H_n$ (where $H_n \in [\ln n, \ln n + 1]$ is the nth harmonic number), which is optimal to constant factors [5]. For the "unfair" setting, however, the competitive ratio does not decrease as substantially with r as we would like: the ratio becomes $(1 + 1/r)H_n$. Therefore, the partitioning bound resulting from Theorem 1 is worse than for other algorithms; specifically, the coefficient on L is not a constant, even for large values of r.

Corollary 2 *The loss of* Marking *for n experts will be at most*

$$H_n(1 + 1/r)L + H_n(r + 1)k + r$$

for any k-segment partition of loss L.

In fact, it is easy to construct examples where the loss of Marking is $\Omega(L \log n)$, where L is the loss of the best single expert (i.e., $k = 1$).

In contrast, we describe below an algorithm whose competitive ratio is $1 + O\left(\log(n)/r\right)$, resulting in a much better Experts-DTF partitioning bound.

4.4. Odd-Exponent

The following algorithm of Bartal *et al* [3] for an n-state MTS is inspired by the Linear 2-state algorithm.

Algorithm Odd-Exponent *[3].* Let t be an odd integer, and let $\hat{L}_i$ represent the reduced loss of expert i. Then place

$$p_i = \frac{1}{n} + \frac{1}{n}\sum_{j=1}^{n}\left(\frac{\hat{L}_j - \hat{L}_i}{r}\right)^t$$

probability on expert i.

The motivation for this algorithm is that for $t = 1$ we have a direct generalization of Linear. However, setting $t = 1$ causes the algorithm to spend too much for movement. Larger values of t allow for substantially decreased movement costs at the expense of relatively small increases in task-processing cost. Specifically, the following is a theorem of Bartal *et al* [3].

THEOREM 7 ([3]) *For an n-node uniform task system,* Odd-Exponent *has an r-unfair competitive ratio of* $1 + \frac{1}{r}(2n^{1/t}t)$.

In particular, notice that, if we let t be the odd integer closest to $\ln n$, then the bound provided by Theorem 7 is approximately $1 + \frac{1}{r}(2e \ln n)$.

Theorems 1 and 7 immediately imply a bound for the Experts-DTF setting.

COROLLARY 3 *Let c be a positive real number. Configure* Odd-Exponent *with t the odd integer closest to* $\ln n$ *and* $r = (2n^{1/t}t)c$*, and predict according to its probability distribution. Then the partitioning bound for n experts will be at most*

$$\begin{aligned} &\left(1 + \frac{1}{c}\right) L + \left(2(c+1)n^{1/t}t\right) k + 2cn^{1/t}t \\ \approx\ &\left(1 + \frac{1}{c}\right) L + \left(2e(c+1)\ln n\right) k + 2ec\ln n\ , \end{aligned}$$

for any k-segment partition of loss L.

Proof: This is a corollary of Theorem 7 by applying Theorem 1. The additive term comes from the fact that in the analysis of [3], the algorithm competes against not the minimum reduced loss, but the average, which is at most r more. ∎

4.4.1. Implementation In an implementation of Odd-Exponent, using reduced loss strictly as defined above introduces a problem: the algorithm could allocate negative probability to an expert. (Consider the case where one expert has reduced loss of r while the rest are zero.) The analysis of [3] skirts the issue by observing that we may assume without loss of generality that experts with zero probability incur zero loss, and furthermore, because of the one-step lookahead in the MTS setting, that an expert never receives a greater loss than that needed to set its probability to zero. (An adversary will not choose to give an expert more loss, since the algorithm will not experience it and the loss may hurt the off-line optimal.)

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 reduced loss, $\tilde{L}$, in computing the probability distribution of the strategy. This $\tilde{L}$ is computed as follows. Say the strategy receives a loss vector $\boldsymbol{\ell}$. We will change $\tilde{L}_i$ to become, not $\min\{\tilde{L}_i + \ell_i, \min_j \tilde{L}_j + \ell_j + r\}$ as for reduced loss, but $\min\{\tilde{L}_i + \ell_i, x\}$, where x is the greatest value such that no probabilities are negative. (In an implementation one can compute x using numerical techniques.) This avoids negative probabilities because each probability that would have become negative with the unmodified reduced loss becomes zero instead.

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

5. Multiplicative weight-updating algorithms

As observed in Section 3.1, an analog of Theorem 1 does not hold for Experts-DTF algorithms in the MTS setting, since they may not control their movement costs well. We examine one such algorithm, based on Littlestone and Warmuth's WML, that works well for the Experts-DTF setting but not the MTS setting. We then examine another algorithm, a variant of Herbster and Warmuth's weight-sharing algorithm, that we show succeeds in both settings.

5.1. *A weight-threshold algorithm*

The WML algorithm, due to Littlestone and Warmuth, is the following strategy for the on-line discrete prediction problem.

Algorithm WML *[13].* The algorithm uses two parameters, $\beta \in [0,1]$ and $\alpha \in [0,1/2]$, and maintains a weight w_i for each expert, initialized to 1. When an expert makes a mistake, we multiply its weight w_i by β, but only if its weight is at least $\alpha W/n$, where $W = \sum w_i$. We then select the prediction on which the most weight falls.

The obvious generalization to the Experts-DTF setting is to place w_i/W probability on expert i. However, this does not result in a good worst-case bound. Consider a two-expert example where expert 0 incurs no loss and expert 1 has a loss of one for each trial. Then w_0 remains at one while w_1 stops decreasing when it reaches $\beta\alpha/(2-\beta\alpha)$. For each subsequent trial the algorithm incurs at least $\beta\alpha/2$ loss, so the loss is arbitrarily large.

This motivates a different generalization of the algorithm.

Algorithm Thresh. As in WML, this algorithm uses parameters $\beta \in [0,1]$ and $\alpha \in [0,1/2]$, and maintains a weight w_i on each expert, initialized to 1. When expert i incurs a loss of ℓ_i, we multiply its weight w_i by β^{ℓ_i} but only if its weight is at least $\alpha W/n$, where $W = \sum w_i$. Let $\widehat{W}$ be the sum of weights w_i that are at least $\alpha W/n$. Put zero probability on expert i if $w_i < \alpha W/n$ and put $w_i/\widehat{W}$ probability on expert i otherwise.

5.1.1. Experts-DTF performance

THEOREM 8 *Given n experts,* Thresh *will incur a loss of at most*

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

for any k-segment partition of loss L.

Proof: The proof follows the general structure of the proof in Littlestone and Warmuth [13].

Consider a loss vector $\boldsymbol{\ell}^t$ for trial t. Remembering that $\widehat{W} \geq W(1-\alpha)$, we can bound how the total weight W^t changes after each trial.

$$\begin{aligned} W^{t+1} &= \sum_{w_i^t \geq \alpha W^t/n} \beta^{\ell_i^t} w_i^t + \sum_{w_i^t < \alpha W^t/n} w_i^t \\ &\leq \sum_{w_i^t \geq \alpha W^t/n} \left(1-(1-\beta)\ell_i^t\right) w_i^t + \sum_{w_i^t < \alpha W^t/n} w_i^t \\ &= W^t \left(1-(1-\beta) \sum_{w_i^t \geq \alpha W^t/n} \frac{w_i^t}{W^t}\ell_i^t\right) \\ &\leq W^t \left(1-(1-\beta)(1-\alpha) \sum_{w_i^t \geq \alpha W^t/n} \frac{w_i^t}{\widehat{W}^t}\ell_i^t\right) \\ &= W^t \left(1-(1-\beta)(1-\alpha)\mathbf{p}^t \cdot \boldsymbol{\ell}^t\right) \end{aligned}$$

Consider a segment in which expert i incurs a total loss of L_i. Say that the total weight at the beginning of the segment is W^0. Because Thresh never allows a weight to fall below $\beta\alpha W/n$, the initial weight of expert i in the segment is at least $\beta\alpha W^0/n$. Thus its weight after the segment is at least $\beta^{L_i}\beta\alpha W^0/n$. If we let W^{final} represent the total weight after the segment ends, we can bound our total loss for the segment, $\sum_t \mathbf{p}^t \cdot \boldsymbol{\ell}^t$.

$$\begin{aligned} \frac{\beta^{L_i}\beta\alpha W^0}{n} &\leq W^{final} \\ &\leq W^0 \prod_t \left(1-(1-\beta)(1-\alpha)\mathbf{p}^t \cdot \boldsymbol{\ell}^t\right) \end{aligned}$$

So we have

$$L_i \ln\beta + \ln\frac{\beta\alpha}{n} \leq -(1-\beta)(1-\alpha)\sum_t \mathbf{p}^t \cdot \boldsymbol{\ell}^t ,$$

which gives us a total loss for the segment of

$$\sum_t \mathbf{p}^t \cdot \boldsymbol{\ell}^t \leq \frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)} L_i + \frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)} .$$

Therefore the total cost to Thresh over the partition is at most

$$\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}L + \frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)}k$$

for a k-segment partition of loss L. ■

5.1.2. MTS performance This result, unfortunately, does not carry over to the MTS setting. Consider the two-expert case. Say that expert 1 incurs a large enough loss for its weight to drop to slightly below $\alpha/(2-\alpha)$. At this point, the algorithm has all probability on expert 0. Now, suppose expert 0 incurs a tiny loss, just sufficient to bring w_1 to equal $\alpha W/n$. This forces the algorithm to move $\alpha/2$ probability over to expert 1. Now suppose expert 1 incurs an infinitesimal loss so that $w_1 < \alpha W/n$. This forces the algorithm to move $\alpha W/n$ probability back to expert 0. 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.

(Admittedly, this is a pathological case. Indeed, in the experiments of Section 6, Thresh performs comparably to the Share algorithm we now consider.)

5.2. A weight-sharing experts algorithm

We now describe a weight-sharing algorithm based on the Variable-share algorithm of Herbster and Warmuth [11] and prove that it achieves good bounds in both the Experts-DTF and MTS settings.

Algorithm Share. The algorithm has two parameters: $\beta \in [0,1]$ is the usual penalty parameter and $\alpha \in [0, 1/2]$ is a "sharing" parameter. The algorithm maintains n weights $w_1, \ldots, w_n$, each initialized to 1. The weights are used to determine the probability distribution in the usual way, namely $p_i = w_i/W$, where $W = \sum_i w_i$. Given a loss vector $\boldsymbol{\ell}$ the algorithm updates the weights using the formula:

$$w_i \leftarrow w_i \beta^{\ell_i} + \alpha\Delta/n\,,$$

where $\Delta = \sum_i (w_i - w_i\beta^{\ell_i})$.

The update rule used by this algorithm can be viewed as follows. We first update as usual: $w_i \leftarrow w_i\beta^{\ell_i}$. This reduces the sum of the weights by some amount Δ. We then distribute an α fraction of this Δ evenly among the n experts ($\alpha\Delta/n$ each).

5.2.1. Experts-DTF performance

THEOREM 9 *Given n experts,* Share *incurs a loss of at most*

$$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right)L + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right)k\,,$$

for any k-segment partition of loss L.

Proof: Consider a specific segment of the given sequence of trials. Our goal is to show that, for any expert i, the loss L_A incurred by the algorithm in this segment is at most $(\ln(1/\beta)L_i + \ln(n/\alpha))/(1-\beta)(1-\alpha)$, where L_i is the loss incurred by expert i in the segment.

For time t, define the algorithm's probability distribution to be $\mathbf{p}^t$, the loss vector to be $\boldsymbol{\ell}^t$, and the sum of the weights to be W^t. Then, using the usual weighted-update analysis (e.g., see the first portion of the proof of Theorem 8), we have

$$W^{t+1} \;\leq\; W^t\left(1-(1-\beta)(1-\alpha)\mathbf{p}^t\cdot\boldsymbol{\ell}^t\right) .$$

So, if W^0 is the sum of weights at the start of the segment and W^{final} is the sum of weights at the end of the segment, then W^{final} is bounded by

$$W^{\mathit{final}} \;\leq\; W^0\prod_t\left(1-(1-\beta)(1-\alpha)\mathbf{p}^t\cdot\boldsymbol{\ell}^t\right) . \tag{4}$$

Now consider the weight of expert i. In the update made at time t, the amount added to w_i due to the share update is $\alpha(W^t - W^{t+1})/(1-\alpha)n$. (This is because $W^{t+1} = W^t - \Delta^t + \alpha\Delta^t$, where Δ^t is the value of Δ specified in the algorithm at time t.) In the entire segment, therefore, the total amount added to w_i due to the share updates is $\frac{\alpha(W^0-W^{\mathit{final}})}{(1-\alpha)n}$. Thus, even if w_i is zero at the start of the segment, by the end of the segment we have

$$w_i^{\mathit{final}} \;\geq\; \beta^{L_i}\left(\frac{\alpha(W^0-W^{\mathit{final}})}{(1-\alpha)n}\right) , \tag{5}$$

since the worst case for w_i is if the penalty for the expert's loss applies maximally to the required shared amount, when the penalties all come after the sharing.

For convenience, define

$$\Pi = \prod_t\left(1-(1-\beta)(1-\alpha)\mathbf{p}^t\cdot\boldsymbol{\ell}^t\right) , \tag{6}$$

So inequality (4) can be written as $W^{\mathit{final}} \leq W^0\Pi$. Using the fact that $W^{\mathit{final}} \geq w_i^{\mathit{final}}$, plugging into inequality (5) we get:

$$\Pi \;\geq\; \frac{\beta^{L_i}\alpha(1-\Pi)}{(1-\alpha)n} .$$

We can now solve for Π.

$$\Pi \;\geq\; \frac{\beta^{L_i}\alpha}{(1-\alpha)n+\beta^{L_i}\alpha} \geq \frac{\beta^{L_i}\alpha}{n}$$

This gives us

$$-\ln\Pi \leq L_i\ln(1/\beta) + \ln\left(\frac{n}{\alpha}\right) .$$

Recalling the definition of Π in (6), we notice that

$$-\ln \Pi \geq (1-\beta)(1-\alpha)L_A \ ,$$

so

$$L_A \leq \frac{L_i \ln(1-\beta) + \ln(n/\alpha)}{(1-\beta)(1-\alpha)}$$

as desired. ■

For $\alpha = 1/n$, in the limit as $\beta \to 1$ and $n \to \infty$, the bound in Theorem 9 approaches roughly $(1+\epsilon/2)L + (\frac{2}{\epsilon}\ln n)k$, for $\epsilon = 1-\beta$. Comparing this to the bound in Corollary 3 for Odd-Exponent, we see that for the same L coefficient (setting $\frac{1}{c} = \frac{2}{\epsilon}$) the bound on Share has a k coefficient that is approximately $2e$ times better. Of course, the bound in Corollary 3 may not be tight, both because the bound in Theorem 7 is not known to be tight for the MTS setting and because the conversion of Theorem 1 may lose some exactness.

5.2.2. MTS performance As we have already remarked, the Share algorithm performs well in both settings. The bound for the MTS setting is almost as good as that of Odd-Exponent. The primary loss is the appearance of a new $(\log r)/r$ term in the competitive ratio.

THEOREM 10 *Given r, configure* Share *with $\alpha = 1/(2r+1)$ and $\beta = \max\{1/2, 1-\gamma\}$, where $\gamma = \frac{1}{r}\ln(n/\alpha)$. Then in the uniform MTS setting,* Share *has an r-unfair competitive ratio of at most*

$$1 + \frac{8}{r}\left(\ln n + \ln(2r+1)\right) \ .$$

(Note: these constants have not been optimized; the formulas used for α and β are designed mainly to simplify the analysis.)

Proof: Suppose that the optimal off-line strategy makes k moves on the given task sequence. Let OPT^t denote the cost incurred by this strategy between its $(t-1)$st and tth moves, and let $\mathit{localcost}_{\mathrm{OPT}} = \sum_{t=1}^{k+1}\mathrm{OPT}^t$. Then we can write the optimal off-line cost $\mathit{cost}_{\mathrm{OPT}}$ as

$$\mathit{cost}_{\mathrm{OPT}} = \mathit{localcost}_{\mathrm{OPT}} + rk \ .$$

Let us consider separately the local cost *localcost* and the movement cost *movecost* of the Share algorithm. Theorem 9 shows that the local cost satisfies

$$\mathit{localcost} \ \leq \ \mathit{localcost}_{\mathrm{OPT}}\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) + k\left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right) . \qquad (7)$$

(In fact, the MTS problem has one-step lookahead, which only helps.) Define ϵ as $1-\beta$, so $\epsilon = \min\{1/2, \gamma\}$. In the case that $\epsilon = \gamma$, by inequality (7) we have

$$\begin{aligned} localcost &\leq localcost_{\mathrm{OPT}}(1+\epsilon/2+\epsilon^2/3+\cdots)/(1-\alpha) + kr/(1-\alpha) \\ &\leq localcost_{\mathrm{OPT}}(1+\epsilon)/(1-\alpha) + kr/(1-\alpha) \\ &\leq cost_{\mathrm{OPT}}(1+\gamma)/(1-\alpha)\,. \end{aligned}$$

On the other hand, if $\epsilon = 1/2$, we have $\gamma > 1/2$ and hence

$$\begin{aligned} localcost &\leq localcost_{\mathrm{OPT}}(1+\epsilon)/(1-\alpha) + 2kr\gamma/(1-\alpha) \\ &\leq cost_{\mathrm{OPT}}(1+2\gamma)/(1-\alpha)\,. \end{aligned}$$

Now we would like to show that the movement cost is bounded by $(3\epsilon/2)localcost$. If we can get this, then we can bound our total cost by

$$\begin{aligned} cost_{\mathrm{OPT}}(1+2\gamma)(1+3\epsilon/2)/(1-\alpha) &< cost_{\mathrm{OPT}}(1+5\gamma)/(1-\alpha) \\ &< cost_{\mathrm{OPT}}(1+8\gamma)\,, \end{aligned}$$

which is our goal.

To analyze the movement cost, note that the total weight W' after the weight update is less than W, and so we have the following.

$$\begin{aligned} d(\mathbf{p},\mathbf{p}') &= \sum_{i:p_i>p'_i}\left(\frac{w_i}{W} - \frac{w_i\beta^{\ell_i}+\alpha\Delta/n}{W'}\right) \\ &\leq \sum_{i:p_i>p'_i}\left(\frac{w_i}{W} - \frac{w_i\beta^{\ell_i}}{W'}\right) \\ &\leq \sum_{i:p_i>p'_i}\left(\frac{w_i}{W} - \frac{w_i\beta^{\ell_i}}{W}\right) \end{aligned}$$

From here we can proceed as in Theorem 3 to bound the movement cost by $\ln(1/\beta) \leq 3\epsilon/2$ times the local cost. ∎

6. Process migration simulation

We now describe some brief experimental results comparing the algorithms analyzed above and others on data representing a process migration problem. Process migration has aspects of both the MTS problem and the Experts-DTF settings. There is a cost to move between machines, but there is also zero lookahead.

As data for process migration, we use 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 $\ell_i^t = 1$ if the machine had a large load average (more than 0.5), and $\ell_i^t = 0$ if it had a small load average. The intent of this is to model the decision faced by a

Table 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		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

"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 [8]. 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.

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 Marking, Odd-Exponent (with $t = 3$), Thresh, and Share. 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 for real problems 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 included the expected number of moves.

This data is summarized in Table 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 2 summarizes these results. For an inter-machine distance of 1.0, the optimal off-line sequence suffered a loss

Table 2. Performance relative to optimal off-line sequence ($d = 1.0$) 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 3. Summary of theoretical results

algorithm	competitive ratio		partitioning bound	
Linear ($n = 2$)	$1 + \frac{1}{r}$	[4]	$\left(1 + \frac{1}{2r}\right) L + \left(r + \frac{1}{2}\right) k$	(Th 6)
Marking	$H_n \left(1 + \frac{1}{r}\right)$	[5]	$H_n \left(1 + \frac{1}{r}\right) L + H_n(r+1)k$	(Cor 2)
Odd-Exponent	$1 + \frac{2e}{r} \ln(n)$	[3]	$\left(1 + \frac{1}{c}\right) L + (2e(r+1)\ln n)\, k$	(Cor 3)
Thresh	unbounded		$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) L + \left(\frac{\ln(n/\beta\alpha)}{(1-\beta)(1-\alpha)}\right) k$	(Th 8)
Share	$1 + \frac{8}{r} \ln(n(2r+1))$	(Th 10)	$\left(\frac{\ln(1/\beta)}{(1-\beta)(1-\alpha)}\right) L + \left(\frac{\ln(n/\alpha)}{(1-\beta)(1-\alpha)}\right) k$	(Th 9)

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.

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.

One of the most fascinating features of this study is how similarly these algorithms perform despite the differences in their techniques and backgrounds. The experiments support the analysis of this paper, which illustrates how these separate techniques can work for both scenarios.

7. Summary

In this paper we consider the relationship between the Metrical Task System problem and the problem of tracking the best expert in the "decision-theoretic" experts setting. We show that any MTS algorithm also has a partitioning bound in the Experts-DTF setting based on its r-unfair competitive ratio in the uniform metric space. We also show how several specific Experts-DTF algorithms can be applied to the MTS setting, and introduce and analyze several new variations. Their bounds are summarized in Table 3. As a special case of our results, we show how the basic randomized Weighted-Majority algorithm can be used for the problem of combining on-line algorithms on-line.

Interestingly, algorithms of quite different styles — the "work-function-based" or "reduced-loss-based" algorithms designed for the MTS problem, and the multiplicative weight-updating schemes more common in the Experts-DTF setting — yield similar bounds. In the MTS setting, the somewhat better theoretical bounds of Odd-Exponent may be offset by the relative simplicity, intuitiveness, and ease of implementation of Share.

There are many settings that combine the problem of picking the best decision/expert/state with a cost for frequently changing one's mind. The process migration problem is one for which we have performed some simple experiments. It would be interesting to see more generally what sorts of algorithms turn out to be best for other problems of this nature.

Notes

1. Those papers used a definition of *r-unfair* in which OPT pays the *same* as the on-line algorithm for movement, but OPT pays r times *less* locally. The motivation was that each "state" was really a sub-space in which (recursively) one had an on-line algorithm with competitive ratio r. The definition we are using is the same as that one, scaled by a factor of r. We choose to define it as we do here to make the connection to the Experts-DTF setting more clear.
2. This can be thought of as an ∞-unfair MTS problem.
3. Azar *et al* [1] do not make this assumption. In their formulation, to move from the state of A_i to the state of A_j, one might potentially have to undo all of A_i's actions since the beginning of time and redo all of A_j's. This is the reason for their worse bounds.
4. If the losses are always in $\{0, 1\}$ then the proof can be simplified by ignoring δ (it will always be 0) and ignoring cases 2 and 4 (which only occur when p_0 or p_1 is 0).

References

1. 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.
2. 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.

3. 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.
4. 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.
5. A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *J of the ACM*, 39(4):745–763, 1992.
6. N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, D. Haussler, R.E. Schapire, and M.K. Warmuth. How to use expert advice. In *Proc ACM Symposium on Theory of Computing*, pages 382–391, 1993.
7. T. Chung. Approximate methods for sequential decision making using expert advice. In *Proc ACM Workshop on Computational Learning Theory*, pages 183–189. ACM Press, New York, NY, 1994.
8. M. Eskicioglu. Process migration in distributed systems: A comparitive survey. Technical Report TR 90-3, University of Alberta, January 1990.
9. A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *J of Algorithms*, 12:685–699, 1991.
10. 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.
11. M. Herbster and M. Warmuth. Tracking the best expert. *J Machine Learning*, 32(2):286–294, 1998.
12. S. Irani and S. Seiden. Randomized algorithms for metrical task systems. *Theoretical Computer Science*, 194(1–2):163–182, 1998.
13. N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
14. S. Seiden. Unfair problems and randomized algorithms for metrical task systems. *Information and Computation*, 1998. To appear.