Journal of Machine Learning Research 2009 (10) 1687-1718 Submitted 9/08; Published 7/09

Learning Permutations with Exponential Weights*

David P. Helmbold DPH@CSE.UCSC.EDU
Manfred K. Warmuth† MANFRED@CSE.UCSC.EDU
Computer Science Department
University of California, Santa Cruz
Santa Cruz, CA 95064

Editor: Yoav Freund

Abstract

We give an algorithm for the on-line learning of permutations. The algorithm maintains its uncertainty about the target permutation as a doubly stochastic weight matrix, and makes predictions using an efficient method for decomposing the weight matrix into a convex combination of permutations. The weight matrix is updated by multiplying the current matrix entries by exponential factors, and an iterative procedure is needed to restore double stochasticity. Even though the result of this procedure does not have a closed form, a new analysis approach allows us to prove an optimal (up to small constant factors) bound on the regret of our algorithm. This regret bound is significantly better than that of either Kalai and Vempala's more efficient Follow the Perturbed Leader algorithm or the computationally expensive method of explicitly representing each permutation as an expert.

Keywords: NEED KEYWORDS – lower case, separated by comma, no period

1. Introduction

Finding a good permutation is a key aspect of many problems such as the ranking of search results or matching workers to tasks. In this paper we present an efficient and effective on-line algorithm for learning permutations in a model related to the on-line allocation model of learning with experts (Freund and Schapire, 1997). In each trial, the algorithm probabilistically chooses a permutation and then incurs a linear loss based on how appropriate the permutation was for that trial. The *regret* is the total expected loss of the algorithm on the whole sequence of trials minus the total loss of the best permutation chosen in hindsight for the whole sequence, and the goal is to find algorithms that have provably small worst-case regret.

For example, one could consider a commuter airline which owns n airplanes of various sizes and flies n routes.[1] Each day the airline must match airplanes to routes. If too small an airplane is assigned to a route then the airline will loose revenue and reputation due to unserved potential passengers. On the other hand, if too large an airplane is used on a long route then the airline could have larger than necessary fuel costs. If the number of passengers wanting each flight were known ahead of time, then choosing an assignment is a weighted matching problem. In the on-line allocation model, the airline first chooses a distribution over possible assignments of airplanes to

*. An earlier version of this paper appears in *Proceedings of the Twentieth Annual Conference on Computational Learning Theory* (COLT 2007), published by Springer as LNAI 4539.
†. Manfred K. Warmuth acknowledges the support of NSF grant IIS 0325363.
1. We assume that each route starts and ends at the airline's home airport.

routes and then randomly selects an assignment from the distribution. The *regret* of the airline is the earnings of the single best assignment for the whole sequence of passenger requests minus the total expected earnings of the on-line assignments. When airplanes and routes are each numbered from 1 to n, then an assignment is equivalent to selecting a permutation. The randomness helps protect the on-line algorithm from adversaries and allows one to prove good bounds on the algorithm's regret for arbitrary sequences of requests.

Since there are $n!$ permutations on n elements, it is infeasible to simply treat each permutation as an expert and apply one of the expert algorithms that uses exponential weights. Previous work has exploited the combinatorial structure of other large sets of experts to create efficient algorithms (see Helmbold and Schapire, 1997; Takimoto and Warmuth, 2003; Warmuth and Kuzmin, 2008, for examples). Our solution is to make a simplifying assumption on the loss function which allows the new algorithm, called PermELearn, to maintain a sufficient amount of information about the distribution over $n!$ permutations while using only n^2 weights.

We represent a permutation of n elements as an $n \times n$ *permutation matrix* Π where $\Pi_{i,j} = 1$ if the permutation maps element i to position j and $\Pi_{i,j} = 0$ otherwise. As the algorithm randomly selects a permutation $\widehat{\Pi}$ at the beginning of a trial, an adversary simultaneously selects an arbitrary *loss matrix* $L \in [0,1]^{n \times n}$ which specifies the loss of all permutations for the trial. Each entry $L_{i,j}$ of the loss matrix gives the loss for mapping element i to j, and the loss of any whole permutation is the sum of the losses of the permutation's mappings, that is, the loss of permutation Π is $\sum_i L_{i,\Pi(i)} = \sum_{i,j} \Pi_{i,j} L_{i,j}$. Note that the per-trial expected losses can be as large as n, as opposed to the common assumption for the expert setting that the losses are bounded in $[0,1]$. In Section 3 we show how a variety of intuitive loss motifs can be expressed in this matrix form.

This assumption that the loss has a linear matrix form ensures the expected loss of the algorithm can be expressed as $\sum_{i,j} W_{i,j} L_{i,j}$, where $W = \mathbb{E}(\widehat{\Pi})$. This expectation W is an $n \times n$ *weight matrix* which is *doubly stochastic*, that is, it has non-negative entries and the property that every row and column sums to 1. The algorithm's uncertainty about which permutation is the target is summarized by W; each weight $W_{i,j}$ is the probability that the algorithm predicts with a permutation mapping element i to position j. It is worth emphasizing that the W matrix is only a *summary* of the distribution over permutations used by any algorithm (it doesn't indicate which permutations have non-zero probability, for example). However, this summary is *sufficient* to determine the algorithm's expected loss when the losses of permutations have the assumed loss matrix form.

Our PermELearn algorithm stores the weight matrix W and must convert W into an efficiently sampled distribution over permutations in order to make predictions. By Birkhoff's Theorem, every doubly stochastic matrix can be expressed as the convex combination of at most $n^2 - 2n + 2$ permutations (see, e.g., Bhatia, 1997). In Appendix A we show that a greedy matching-based algorithm efficiently decomposes any doubly stochastic matrix into a convex combination of at most $n^2 - 2n + 2$ permutations. Although the efficacy of this algorithm is implied by standard dimensionality arguments, we give a new combinatorial proof that provides independent insight as to why the algorithm finds a convex combination matching Birkhoff's bound. Our algorithm for learning permutations predicts with a random $\widehat{\Pi}$ sampled from the convex combination of permutations created by decomposing weight matrix W. It has been applied recently for pricing combinatorial markets when the outcomes are permutations of objects (Chen et al., 2008).

The PermELearn algorithm updates the entries of its weight matrix using exponential factors commonly used for updating the weights of experts in on-line learning algorithms (Littlestone and Warmuth, 1994; Vovk, 1990; Freund and Schapire, 1997): each entry $W_{i,j}$ is multiplied by a factor

$e^{-\eta L_{i,j}}$. Here η is a positive learning rate that controls the "strength" of the update (When $\eta = 0$, than all the factors are one and the update is vacuous). After this update, the weight matrix no longer has the doubly stochastic property, and the weight matrix must be projected back into the space of doubly stochastic matrices (called "Sinkhorn balancing", see Section 4) before the next prediction can be made.

In Theorem 4 we bound the expected loss of PermELearn over any sequence of trials by

$$\frac{n\ln n + \eta \mathcal{L}_{\text{best}}}{1 - e^{-\eta}}, \tag{1}$$

where n is the number of elements being permuted, η is the learning rate, and $\mathcal{L}_{\text{best}}$ is the loss of the best permutation on the entire sequence. If an upper bound $\mathcal{L}_{\text{est}} \geq \mathcal{L}_{\text{best}}$ is known, then η can be tuned (as in Freund and Schapire, 1997) and the expected loss bound becomes

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}} n \ln n} + n \ln n, \tag{2}$$

giving a bound of $\sqrt{2\mathcal{L}_{\text{est}} n \ln n} + n \ln n$ on the worst case expected regret of the tuned PermELearn algorithm. We also prove a matching lower bound (Theorem 6) of $\Omega(\sqrt{\mathcal{L}_{\text{best}} n \ln n})$ for the expected regret of any algorithm solving our permutation learning problem.

A simpler and more efficient algorithm than PermELearn maintains the sum of the loss matrices on the the previous trials. Each trial it adds random perturbations to the cumulative loss matrix and then predicts with the permutation having minimum perturbed loss. This "Follow the Perturbed Leader" algorithm (Kalai and Vempala, 2005) has good regret bounds for many on-line learning settings. However, the regret bound we can obtain for it in the permutation setting is about a factor of n worse than the bound for PermELearn and the lower bound.

Although computationally expensive, one can also consider running the Hedge algorithm while explicitly representing each of the $n!$ permutations as an expert. If T is the sum of the loss matrices over the past trials and F is the $n \times n$ matrix with entries $F_{i,j} = e^{-\eta T_{i,j}}$, then the weight of each permutation expert Π is proportional to the product $\prod_i F_{i,\Pi(i)}$ and the normalization constant is the permanent of the matrix F. Calculating the permanent is a known #P-complete problem and sampling from this distribution over permutations is very inefficient (Jerrum et al., 2004). Moreover since the loss range of a permutation is $[0, n]$, the standard loss bound for the algorithm that uses one expert per permutation must be scaled up by a factor of n, becoming

$$\mathcal{L}_{\text{best}} + n\sqrt{2\frac{\mathcal{L}_{\text{est}}}{n}\ln(n!)} + n\ln(n!) \approx \mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}} n^2 \ln n} + n^2 \ln n.$$

This expected loss bound is similar to our expected loss bound for PermELearn in Equation (2), except that the $n \ln n$ terms are replaced by $n^2 \ln n$. Our method based on Sinkhorn balancing bypasses the estimation of permanents and somehow PermELearn's implicit representation and prediction method exploit the structure of permutations and lets us obtain the improved bound. We also give a matching lower bound that shows PermELearn has the optimum regret bound (up to a small constant factor). It is an interesting open question whether the structure of permutations can be exploited to prove bounds like (2) for the Hedge algorithm with one expert per permutation.

PermELearn's weight updates belong to the Exponentiated Gradient family of updates (Kivinen and Warmuth, 1997) since the components $L_{i,j}$ of the loss matrix that appear in the exponential factor are the derivatives of our linear loss with respect to the weights $W_{i,j}$. This family of updates usually maintains a probability vector as its weight vector. In that case the normalization of

the weight vector is straightforward and is folded directly into the update formula. Our new algorithm PermELearn for learning permutations maintains a doubly stochastic matrix with n^2 weights. The normalization alternately normalizes the rows and columns of the matrix until convergence (Sinkhorn balancing). This may require an unbounded number of steps and the resulting matrix does not have a closed form. Despite this fact, we are able to prove bounds for our algorithm.

We first show that our update minimizes a tradeoff between the loss and a relative entropy between doubly stochastic matrices. This relative entropy becomes our measure of progress in the analysis. Luckily, the un-normalized multiplicative update already makes enough progress (towards the best permutation) to achieve the loss bound quoted above. Finally, we interpret the iterations of Sinkhorn balancing as Bregman projections with respect to the same relative entropy and show using the properties of Bregman projections that these projections can only increase the progress and thus don't hurt the analysis (Herbster and Warmuth, 2001).

Our new insight of splitting the update into an un-normalized step followed by a normalization step also leads to a streamlined proof of the loss bound for the Hedge algorithm in the standard expert setting that is interesting in its own right. Since the loss in the allocation setting is linear, the bounds can be proven in many different ways, including potential based methods (see, e.g., Kivinen and Warmuth, 1999; Gordon, 2006; Cesa-Bianchi and Lugosi, 2006). For the sake of completeness we reprove our main loss bound for PermELearn using potential based methods in Appendix B. We show how potential based proof methods can be extended to handle linear equality constraints that don't have a solution in closed form, paralleling a related extension to linear *in*equality constraints in Kuzmin and Warmuth (2007). In this appendix we also discuss the relationship between the projection and potential based proof methods. In particular, we show how the Bregman projection step corresponds to plugging in suboptimal dual variables into the potential.

The remainder of the paper is organized as follows. We introduce our notation in the next section. Section 3 presents the permutation learning model and gives several intuitive examples of appropriate loss motifs. Section 4 gives the PermELearn algorithm and discusses its computational requirements. One part of the algorithm is to decompose the current doubly stochastic matrix into a small convex combination of permutations using a greedy algorithm. The bound on the number of permutations needed to decompose the weight matrix is deferred to Appendix A. We then bound PermELearn's regret in Section 5 in a two-step analysis that uses a relative entropy as a measure of progress. To exemplify the new techniques, we also analyze the basic Hedge algorithm with the same methodology. The regret bounds for Hedge and PermELearn are re-proven in Appendix B using potential based methods. In Section 6, we apply the "Follow the Perturbed Leader" algorithm to learning permutations and show that the resulting regret bounds are not as good. In Section 7 we prove a lower bound on the regret when learning permutations that is within a small constant factor of our regret bound on the tuned PermELearn algorithm. The concluding section describes extensions and directions for further work.

2. Notation

All matrices will be $n \times n$ matrices. When A is a matrix, $A_{i,j}$ denotes the entry of A in row i, and column j. We use $A \bullet B$ to denote the dot product between matrices A and B, that is, $\sum_{i,j} A_{i,j} B_{i,j}$. We use single superscripts (e.g., A^k) to identify matrices/permutations from a sequence.

Permutations on n elements are frequently represented in two ways: as a bijective mapping of the elements $\{1,\ldots,n\}$ into the positions $\{1,\ldots,n\}$ or as a permutation matrix which is an $n \times n$

binary matrix with exactly one "1" in each row and each column. We use the notation Π (and $\widehat{\Pi}$) to represent a permutation in either format, using the context to indicate the appropriate representation. Thus, for each $i \in \{1,\ldots,n\}$, we use $\Pi(i)$ to denote the position that the ith element is mapped to by permutation Π, and matrix element $\Pi_{i,j} = 1$ if $\Pi(i) = j$ and 0 otherwise.

If L is a matrix with n rows then the product ΠL permutes the rows of L:

$$\Pi = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad L = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix} \qquad \Pi L = \begin{pmatrix} 21 & 22 & 23 & 24 \\ 41 & 42 & 43 & 44 \\ 31 & 32 & 33 & 34 \\ 11 & 12 & 13 & 14 \end{pmatrix}.$$

perm. $(2,4,3,1)$ as matrix — an arbitrary matrix — permuting the rows

Convex combinations of permutations create *doubly stochastic* or *balanced* matrices: non-negative matrices whose n rows and n columns each sum to one. Our algorithm maintains its uncertainty about which permutation is best as a doubly stochastic weight matrix W and needs to randomly select a permutation from some distribution whose expectation is W. By Birkhoff's Theorem (see, e.g., Bhatia, 1997), for every doubly stochastic matrix W there is a decomposition into a convex combination of at most $n^2 - 2n + 2$ permutation matrices. We show in Appendix A how a decomposition of this size can be found effectively. This decomposition gives a distribution over permutations whose expectation is W that now can be effectively sampled because its support is at most $n^2 - 2n + 2$ permutations.

3. On-line Protocol

We are interested in learning permutations in a model related to the on-line allocation model of learning with experts (Freund and Schapire, 1997). In that model there are N experts and at the beginning of each trial the algorithm allocates a probability distribution w over the experts. The algorithm picks expert i with probability w_i and then receives a loss vector $\ell \in [0,1]^N$. Each expert i incurs loss ℓ_i and the expected loss of the algorithm is $w \cdot \ell$. Finally, the algorithm updates its distribution w for the next trial.

In case of permutations we could have one expert per permutation and allocate a distribution over the $n!$ permutations. Explicitly tracking this distribution is computationally expensive, even for moderate n. As discussed in the introduction, we assume that the losses in each trial can be specified by a loss matrix $L \in [0,1]^{n \times n}$ where the loss of each permutation Π has the linear form $\sum_i L_{i,\Pi(i)} = \Pi \bullet L$. If the algorithm's prediction $\widehat{\Pi}$ is chosen probabilistically in each trial then the algorithm's expected loss is $\mathbb{E}[\widehat{\Pi} \bullet L] = W \bullet L$, where $W = \mathbb{E}[\widehat{\Pi}]$. This expected prediction W is an $n \times n$ doubly stochastic matrix and algorithms for learning permutations under the linear loss assumption can be viewed as implicitly maintaining such a doubly stochastic weight matrix.

More precisely, the on-line algorithm follows the following protocol in each trial:

- The learner (probabilistically) chooses a permutation $\widehat{\Pi}$, and let $W = \mathbb{E}(\widehat{\Pi})$.
- Nature simultaneously chooses a loss matrix $L \in [0,1]^{n \times n}$ for the trial.
- At the end of the trial, the algorithm is given L. The loss of $\widehat{\Pi}$ is $\widehat{\Pi} \bullet L$ and the expected loss of the algorithm is $W \bullet L$.

- Finally, the algorithm updates its distribution over permutations for the next trial, implicitly updating matrix W.

Although our algorithm can handle arbitrary sequences of loss matrices $L \in [0,1]^{n\times n}$, nature could be significantly more restricted. Many ranking applications have an associated loss motif M and nature is constrained to choose (row) permutations of M as its loss matrix L. In effect, at each trial nature chooses a "correct" permutation Π and uses the loss matrix $L = \Pi M$. Note that the permutation left-multiplies the loss motif, and thus permutes the rows of M. If nature chooses the identity permutation then the loss matrix L is the motif M itself. When M is known to the algorithm, it suffices to give the algorithm only the permutation Π at the end of the trial, rather than the loss matrix L itself. Figure 1 gives examples of loss motifs.

The last loss in Figure 1 is related to a competitive List Update Problem where an algorithm services requests to a list of n items. In the List Update Problem the cost of a request is the requested item's current position in the list. After each request, the requested item can be moved forward in the list for free, and additional rearrangement can be done at a cost of one per transposition. The goal is for the algorithm to be cost-competitive with the best static ordering of the elements in hindsight. Note that the transposition cost for additional list rearrangement is not represented in the permutation loss motif. Blum et al. (2003) give very efficient algorithms for the List Update Problem that do not do additional rearranging of the list (and thus do not incur the cost neglect by the loss motif). In our notation, their bound has the same form as ours (1) but with the $n\ln n$ factors replaced by $O(n)$. However, our lower bound (see Section 7) shows that the $n\ln n$ factors in (2) are necessary in the general permutation setting.

Note that many compositions of loss motifs are possible. For example, given two motifs with their associated losses, any convex combination of the motifs creates a new motif for the same convex combination of the associated losses. Other component-wise combinations of two motifs (such as product or max) can also produce interesting loss motifs, but the combination usually cannot be distributed across the matrix dot-product calculation, and so cannot be expressed as a simple linear function of the original losses.

4. PermELearn Algorithm

Our permutation learning algorithm uses exponenential weights and we call it PermELearn. It maintains an $n \times n$ doubly stochastic weight matrix W as its main data structure, where $W_{i,j}$ is the probability that PermELearn predicts with a permutation mapping element i to position j. In the absence of prior information it is natural to start with uniform weights, that is, the matrix with $\frac{1}{n}$ in each entry.

In each trial PermELearn does two things:

1. Choose a permutation $\widehat{\Pi}$ from some distribution such that $\mathbb{E}[\widehat{\Pi}] = W$.

2. Create a new doubly stochastic matrix $\widetilde{W}$ for use in the next trial based on the current weight matrix W and loss matrix L.

loss $\mathcal{L}(\widehat{\Pi},\Pi)$	motif M
the number of elements i where $\widehat{\Pi}(i) \neq \Pi$	$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$
$\frac{1}{n-1}\sum_{i=1}^{n} \lvert\widehat{\Pi}(i) - \Pi(i)\rvert$, how far the elements are from their "correct" positions (the division by $n-1$ ensures that the entries of M are in $[0,1]$.)	$\frac{1}{3}\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$
$\frac{1}{n-1}\sum_{i=1}^{n} \frac{\lvert\widehat{\Pi}(i)-\Pi(i)\rvert}{\Pi(i)}$, a position weighted version of the above emphasizing the early positions in Π	$\frac{1}{3}\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1/2 & 0 & 1/2 & 1 \\ 2/3 & 1/3 & 0 & 1/3 \\ 3/4 & 1/2 & 1/4 & 0 \end{pmatrix}$
the number of elements mapped to the first half by Π but the second half by $\widehat{\Pi}$, or vice versa	$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$
the number of elements mapped to the first two positions by Π that fail to appear in the top three position of $\widehat{\Pi}$	$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
the number of links traversed to find the first element of Π in a list ordered by $\widehat{\Pi}$	$\frac{1}{3}\begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Figure 1: Loss motifs

Choosing a permutation is done by Algorithm 1. The algorithm greedily decomposes W into a convex combination of at most n^2-2n+2 permutations (see Theorem 7), and then randomly selects one of these permutations for the prediction.[2]

Our decomposition algorithm uses a Temporary matrix A initialized to the weight matrix W. Each iteration of Algorithm 1 finds a permutation Π where each $A_{i,\Pi(i)} > 0$. This can be done by finding a perfect matching on the $n \times n$ bipartite graph containing the edge i, j whenever $A_{i,j} > 0$. We shall soon see that each matrix A is a constant times a doubly stochastic matrix, so the existence of a suitable permutation Π follows from Birkhoff's Theorem. Given such a permutation Π, the algorithm updates A to $A - \alpha\Pi$ where $\alpha = \min_i A_{i,\Pi(i)}$. The updated matrix A has non-negative entries and has strictly more zeros than the original A. Since the update decreases each row and

2. The decomposition is usually not unique and the implementation may have a bias as to exactly which convex combination is chosen.

Algorithm 1 PermELearn: Selecting a permutation

Require: a doubly stochastic $n \times n$ matrix W

$A := W$; $q = 0$;
repeat
$q := q + 1$;
Find permutation Π^q such that $A_{i,\Pi^q(i)}$ is positive for each $i \in \{1,\ldots,n\}$
$\alpha_q := \min_i A_{i,\Pi^q(i)}$
$A := A - \alpha_q \Pi^q$
until All entries of A are zero $\quad \{$at end of loop $W = \sum_{k=1}^{q} \alpha_k \Pi^k\}$
Randomly select and return a $\widehat{\Pi} \in \{\Pi^1,\ldots,\Pi^q\}$ using probabilities $\alpha_1,\ldots,\alpha_q$.

Algorithm 2 PermELearn: Weight Matrix Update

Require: learning rate η, loss matrix L, and doubly stochastic weight matrix W

Create W' where each $W'_{i,j} = W_{i,j} e^{-\eta L_{i,j}}$ (3)

Create doubly stochastic $\widetilde{W}$ by re-balancing the rows and columns of W' (Sinkhorn balancing) and update W to $\widetilde{W}$.

column sum by α and the original matrix W was doubly stochastic, each matrix A will have rows and columns that sum to the same amount. In other words, each matrix A created during Algorithm 1 is a constant times a doubly stochastic matrix, and thus (by Birkhoff's Theorem) is a constant times a convex combination of permutations.

After at most $n^2 - n$ iterations the algorithm arrives at a matrix A having exactly n non-zero entries, so this A is a constant times a permutation matrix. Therefore, Algorithm 1 decomposes the original doubly stochastic matrix into the convex combination of (at most) $n^2 - n + 1$ permutation matrices. The more refined arguments in Appendix A shows that the Algorithm 1 never uses more than $n^2 - 2n + 2$ permutations, matching the bound given by Birkhoff's Theorem.

Several improvements are possible. In particular, we need not compute each perfect matching from scratch. If only z entries of A are zeroed by a permutation, then that permutation is still a matching of size $n - z$ in the graph for the updated matrix. Thus we need to find only z augmenting paths to complete the perfect matching. The entire process thus requires finding $O(n^2)$ augmenting paths at a cost of $O(n^2)$ each, for a total cost of $O(n^4)$ to decompose weight matrix W into a convex combination of permutations.

4.1 Updating the Weights

In the second step, Algorithm 2 updates the weight matrix by multiplying each $W_{i,j}$ entry by the factor $e^{-\eta L_{i,j}}$. These factors destroy the row and column normalization, so the matrix must be re-balanced to restore the doubly-stochastic property. There is no closed form for the normalization step. The standard iterative re-balancing method for non-negative matrices is called *Sinkhorn balancing*. This method first normalizes each row of the matrix to sum to one, and then normalizes the columns. Since normalizing the columns typically destroys the row normalization, the process must be iterated until convergence (Sinkhorn, 1964).

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix} \quad \xrightarrow{\text{Sinkhorn balancing}} \quad \begin{pmatrix} \frac{\sqrt{2}}{1+\sqrt{2}} & \frac{1}{1+\sqrt{2}} \\ \frac{1}{1+\sqrt{2}} & \frac{\sqrt{2}}{1+\sqrt{2}} \end{pmatrix}$$

Figure 2: Example where Sinkhorn balancing requires infinitely many steps.

Normalizing the rows corresponds to pre-multiplying by a diagonal matrix. The product of these diagonal matrices thus represents the combined effect of the multiple row normalization steps. Similarly, the combined effect of the column normalization steps can be represented by post-multiplying the matrix by a diagonal matrix. Therefore we get the well known fact that Sinkhorn balancing a matrix A results in a doubly stochastic matrix RAC where R and C are diagonal matrices. Each entry $R_{i,i}$ is the positive multiplier applied to row i, and each entry $C_{j,j}$ is the positive multiplier of column j needed to convert A into a doubly stochastic matrix.

In Figure 2 we give a rational matrix that balances to an irrational matrix. Since each row and column balancing step creates rationals, Sinkhorn balancing produces irrationals only in the limit (after infinitely many steps). Multiplying a weight matrix from the left and/or right by non-negative diagonal matrices (e.g., row or column normalization) preserves the ratio of product weights between permutations. That is if $A' = RAC$, then for any two permutations Π_1 and Π_2,

$$\frac{\prod_i A'_{i,\Pi_1(i)}}{\prod_i A'_{i,\Pi_2(i)}} = \frac{\prod_i A_{i,\Pi_1(i)} R_{i,i} C_{\Pi_1(i),\Pi_1(i)}}{\prod_i A_{i,\Pi_2(i)} R_{i,i} C_{\Pi_2(i),\Pi_2(i)}} = \frac{\prod_i A_{i,\Pi_1(i)}}{\prod_i A_{i,\Pi_2(i)}}.$$

Therefore $\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1 \end{pmatrix}$ must balance to a doubly stochastic matrix $\begin{pmatrix} a & 1-a \\ 1-a & a \end{pmatrix}$ such that the ratio of the product weight between the two permutations $(1,2)$ and $(2,1)$ is preserved. This means $\frac{1/2}{1/4} = \frac{a^2}{(1-a)^2}$ and thus $a = \frac{\sqrt{2}}{1+\sqrt{2}}$.

This example leads to another important observation: PermELearn's predictions are different than Hedge's when each permutation is treated as an expert. If each permutation is explicitly represented as an expert, then the Hedge algorithm predicts permutation Π with probability proportional to the product weight, $\prod_i e^{-\eta \sum_t L^t_{i,\Pi(i)}}$. However, algorithm PermELearn predicts differently. With the weight matrix in Figure 4.1, Hedge puts probability $\frac{2}{3}$ on permutation $(1,2)$ and probability $\frac{1}{3}$ on permutation $(2,1)$ while PermELearn puts probability $\frac{\sqrt{2}}{1+\sqrt{2}} \approx 0.59$ on permutation $(1,2)$ and probability $\frac{\sqrt{1}}{1+\sqrt{2}} \approx 0.41$ on permutation (2,1).

There has been much written on the balancing of matrices, and we briefly describe only a few of the results here. Sinkhorn showed that this procedure converges and that the RAC balancing of any matrix A into a doubly stochastic matrix is unique (up to canceling multiples of R and C) if it exists[3] (Sinkhorn, 1964).

A number of authors consider balancing a matrix A so that the row and column sums are $1 \pm \varepsilon$. Franklin and Lorenz (1989) show that $O(\text{length}(A)/\varepsilon)$ Sinkhorn iterations suffice, where length(A) is the bit-length of matrix A's binary representation. Kalantari and Khachiyan (1996) show that

3. Some non-negative matrices, like $\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$, cannot be converted into doubly stochastic matrices because of their pattern of zeros. The weight matrices we deal with have strictly positive entries, and thus can always be made doubly stochastic with an RAC balancing.

$O(n^4 \ln \frac{n}{\varepsilon} \ln \frac{1}{\min A_{i,j}})$ operations suffice using an interior point method. Linial et al. (2000) give a preprocessing step after which only $O((n/\varepsilon)^2)$ Sinkhorn iterations suffice. They also present a strongly polynomial time iterative procedure requiring $\tilde{O}(n^7 \log(1/\varepsilon))$ iterations. Balakrishnan et al. (2004) give an interior point method with complexity $O(n^6 \log(n/\varepsilon))$. Finally, Fürer (2004) shows that if the row and column sums of A are $1 \pm \varepsilon$ then every matrix entry changes by at most $\pm n\varepsilon$ when A is balanced to a doubly stochastic matrix.

4.2 Dealing with Approximate Balancing

With slight modifications, Algorithm PermELearn can handle the situation where its weight matrix is imperfectly balanced (and thus not quite doubly stochastic). As before, let W be the fully balanced doubly stochastic weight matrix, but we now assume that only an approximately balanced $\widehat{W}$ is available to predict from. In particular, we assume that each row and column of $\widehat{W}$ sum to $1 \pm \varepsilon$ for some $\varepsilon < \frac{1}{3}$. Let $s \geq 1 - \varepsilon$ be the smallest row or column sum in $\widehat{W}$.

We modify Algorithm 1 in two ways. First, A is initialized to $\frac{1}{s}\widehat{W}$ rather than W. This ensures every row and column in the initial A sums to at least one, to at most $1 + 3\varepsilon$, and at least one row or column sums to exactly 1. Second, the loop exits as soon as A has an all-zero row or column. Since the smallest row or column sum starts at 1, is decreased by α_k each iteration k, and ends at zero, we have that $\sum_{k=1}^{q} \alpha_k = 1$ and the modified Algorithm 1 still outputs a convex combination of permutations $C = \sum_{k=1}^{q} \alpha_k \Pi^k$. Furthermore, each entry $C_{i,j} \leq \frac{1}{s}\widehat{W}_{i,j}$. We now bound the additional loss of this modified algorithm.

Lemma 1 *If the weight matrix $\widehat{W}$ is approximately balanced so each row and column sum is in $1 \pm \varepsilon$ (for $\varepsilon \leq \frac{1}{3}$) then the modified Algorithm 1 has an expected loss $C \bullet L$ at most $3n^3\varepsilon$ greater than the expected loss $W \bullet L$ of the original algorithm that uses the completely balanced doubly stochastic matrix W.*

Proof Let s be the smallest row or column sum in $\widehat{W}$. Since each row and column sum of $\frac{1}{s}\widehat{W}$ lies in $[1, 1 + 3\varepsilon]$, each entry of $\frac{1}{s}\widehat{W}$ is close to the corresponding entry of the fully balanced W. In particular each $\frac{1}{s}\widehat{W}_{i,j} \leq W_{i,j} + 3n\varepsilon$ (Fürer, 2004). This allows us to bound the expected loss when predicting with the convex combination C in terms of the expected loss using a decomposition of the perfectly balanced W:

$$\begin{aligned} C \bullet L &\leq \frac{1}{s}\widehat{W} \bullet L \\ &= \sum_{i,j} \frac{\widehat{W}_{i,j}}{s} L_{i,j} \\ &\leq \sum_{i,j} (W_{i,j} + 3n\varepsilon) L_{i,j} \\ &\leq W \bullet L + 3n^3\varepsilon. \end{aligned}$$

Therefore the extra loss incurred by using a ε-approximately balanced weight matrix at a particular trial is at most $3n^3\varepsilon$, as desired. ■

If in a sequence of T trials the matrices $\widehat{W}$ are $\varepsilon = 1/(3Tn^3)$ balanced (so that each row and column sum is $1 \pm 1/(3Tn^3)$) then Lemma 1 implies that the total additional expected loss for using approximate balancing is at most 1. The algorithm of Balakrishnan et al. (2004) ε-balances a matrix in $O(n^6 \log(n/\varepsilon))$ time (note that this dominates the time for the loss update and constructing the convex combination). This balancing algorithm with $\varepsilon = 1/(3Tn^3)$ together with the modified prediction algorithm give a method requiring $O(Tn^6 \log(Tn))$ total time over the T trials and having a bound of $\sqrt{2\mathcal{L}_{\text{est}} n \ln n} + n \ln n + 1$ on the worst-case regret.

If the number of trials T is not known in advance then setting ε as a function of t can be helpful. A natural choice is $\varepsilon_t = 1/(3t^2n^3)$. In this case the total extra regret for not having perfect balancing is bounded by $\sum_{t=1}^{T} 1/t^2 \leq 5/3$ and the total computation time over the T trials is still bounded by $O(Tn^6 \log(Tn))$.

One might be concerned about the effects of approximate balancing propagating between trials. However this is not an issue. In the following section we show that the loss updates and balancing can be arbitrarily interleaved. Therefore the modified algorithm can either keep a cumulative loss matrix $L^{\leq t} = \sum_{i=1}^{t} L^i$ and create its next $\widehat{W}$ by (approximately) balancing the matrix with entries $\frac{1}{n} e^{-\eta L_{i,j}^{\leq t}}$, or apply the multiplicative updates to the previous approximately balanced $\widehat{W}$.

5. Bounds for PermELearn

Our analysis of PermELearn follows the entropy-based analysis of the exponentiated gradient family of algorithms (Kivinen and Warmuth, 1997). This style of analysis first shows a per-trial progress bound using relative entropy to a comparator as a measure of progress, and then sums this invariant over the trials to bound the expected total loss of the algorithm. We also show that PermELearn's weight update belongs to the exponentiated gradient family of updates (Kivinen and Warmuth, 1997) since it is the solution to a minimization problem that trades of the loss (in this case a linear loss) against a relative entropy regularization.

Recall that the expected loss of PermELearn on a trial is a linear function of its weight matrix W. Therefore the gradient of the loss is independent of the current value of W. This property of the loss greatly simplifies the analysis. Our analysis for this setting provides a good foundation for learning permutation matrices and lays the groundwork for the future study of other permutation loss functions.

We start our analysis with an attempt to mimic the standard analysis (Kivinen and Warmuth, 1997) for the exponentiated gradient family updates which multiply by exponential factors and re-normalize. The per-trial invariant used to analyze the exponentiated gradient family bounds the decrease in relative entropy from any (normalized) vector u to the algorithm's weight vector by a linear combination of the algorithm's loss and the loss of u on the trial. In our case the weight vectors are matrices and we use the following (un-normalized) relative entropy between matrices A and B with non-negative entries:

$$\Delta(A,B) = \sum_{i,j} A_{i,j} \ln \frac{A_{i,j}}{B_{i,j}} + B_{i,j} - A_{i,j} \quad .$$

Note that this is just the sum of the relative entropies between the corresponding rows (or equivalently, between the corresponding columns):

$$\Delta(A,B) = \sum_{i} \Delta(A_{i,\star}, B_{i,\star}) = \sum_{j} \Delta(A_{\star,j}, B_{\star,j})$$

(here $A_{i,\star}$ is the ith row of A and $A_{\star,j}$ is its jth column).

Unfortunately, the lack of a closed form for the matrix balancing procedure makes it difficult to prove bounds on the loss of the algorithm. Our solution is to break PermELearn's update (Algorithm 2) into two steps, and use only the progress made to the intermediate un-balanced matrix in our per-trial bound (8). After showing that balancing to a doubly stochastic matrix only increases the progress, we can sum the per-trial bound to obtain our main theorem.

5.1 A Dead End

In each trial, PermELearn multiplies each entry of its weight matrix by an exponential factor and then uses one additional factor per row and column to make the matrix doubly stochastic (Algorithm 2 described in Section 4.1):

$$\widetilde{W}_{i,j} := r_i c_j W_{i,j} e^{-\eta L_{i,j}} \tag{4}$$

where the r_i and c_j factors are chosen so that all rows and columns of the matrix $\widetilde{W}$ sum to one.

We now show that PermELearn's update (4) gives the matrix A solving the following minimization problem:

$$\begin{array}{l} \operatorname{argmin} \\ \forall i : \sum_j A_{i,j} = 1 \\ \forall j : \sum_i A_{i,j} = 1 \end{array} \quad (\Delta(A,W) + \eta\,(A \bullet L)). \tag{5}$$

Since the linear constraints are feasible and the divergence is strictly convex, there always is a unique solution, even though the solution does not have a closed form.

Lemma 2 *PermELearn's updated weight matrix* $\widetilde{W}$ (4) *is the solution of* (5).

Proof We form a Lagrangian for the optimization problem:

$$l(A,\rho,\gamma) = \Delta(A,W) + \eta\,(A \bullet L) + \sum_i \rho_i (\sum_j A_{i,j} - 1) + \sum_j \gamma_j (\sum_i A_{i,j} - 1).$$

Setting the derivative with respect to $A_{i,j}$ to 0 yields $A_{i,j} = W_{i,j} e^{-\eta L_{i,j}} e^{-\rho_i} e^{-\gamma_j}$. By enforcing the row and column sum constraints we see that the factors $r_i = e^{-\rho_i}$ and $c_j = e^{-\gamma_j}$ function as row and column normalizers, respectively. ■

We now examine the progress $\Delta(U,W) - \Delta(U,\widetilde{W})$ towards an arbitrary stochastic matrix U. Using Equation (4) and noting that all three matrices are doubly stochastic (so their entries sum to n), we see that

$$\Delta(U,W) - \Delta(U,\widetilde{W}) = -\eta\, U \bullet L + \sum_i \ln r_i + \sum_j \ln c_j.$$

Making this a useful invariant requires lower bounding the sums on the rhs by a constant times $W \bullet L$, the loss of the algorithm. Unfortunately we are stuck because the r_i and c_j normalization factors don't even have a closed form.

5.2 Successful Analysis

Our successful analysis splits the update (4) into two steps:

$$W'_{i,j} := W_{i,j} e^{-\eta L_{i,j}} \quad \text{and} \quad \widetilde{W}_{i,j} := r_i c_j W'_{i,j}, \tag{6}$$

where (as before) r_i and c_j are chosen so that each row and column of the matrix $\widetilde{W}$ sum to one. Using the Lagrangian (as in the proof of Lemma 2), it is easy to see that these W' and $\widetilde{W}$ matrices solve the following minimization problems:

$$W' = \operatorname*{argmin}_A \left(\Delta(A,W) + \eta\,(A \bullet L)\right) \quad \text{and} \quad \widetilde{W} := \operatorname*{argmin}_{\substack{\forall i : \sum_j A_{i,j} = 1 \\ \forall j : \sum_i A_{i,j} = 1}} \Delta(A,W'). \tag{7}$$

The second problem shows that the doubly stochastic matrix $\widetilde{W}$ is the projection of W' onto to the linear row and column sum constraints. The strict convexity of the relative entropy between non-negative matrices and the feasibility of the linear constraints ensure that the solutions for both steps are unique.

We now lower bound the progress $\Delta(U,W) - \Delta(U,W')$ in the following lemma to get our per-trial invariant.

Lemma 3 *For any $\eta > 0$, any doubly stochastic matrices U and W and any trial with loss matrix $L \in [0,1]^{n \times n}$*

$$\Delta(U,W) - \Delta(U,W') \geq (1 - e^{-\eta})(W \bullet L) - \eta(U \bullet L),$$

where W' is the unbalanced intermediate matrix (6) *constructed by PermELearn from W.*

Proof The proof manipulates the difference of relative entropies and uses the inequality $e^{-\eta x} \leq 1 - (1 - e^{-\eta})x$, which holds for any η and any $x \in [0,1]$:

$$\begin{aligned}
\Delta(U,W) - \Delta(U,W') &= \sum_{i,j} \left(U_{i,j} \ln \frac{W'_{i,j}}{W_{i,j}} + W_{i,j} - W'_{i,j} \right) \\
&= \sum_{i,j} \left(U_{i,j} \ln(e^{-\eta L_{i,j}}) + W_{i,j} - W_{i,j} e^{-\eta L_{i,j}} \right) \\
&\geq \sum_{i,j} \left(-\eta L_{i,j} U_{i,j} + W_{i,j} - W_{i,j}(1 - (1 - e^{-\eta}) L_{i,j}) \right) \\
&= -\eta (U \bullet L) + (1 - e^{-\eta})(W \bullet L).
\end{aligned}$$

■

Relative entropy is a Bregman divergence, so the Generalized Pythagorean Theorem (Bregman, 1967) applies. Specialized to our setting, this theorem states that if S is a closed convex set containing some matrix U with non-negative entries, W' is any matrix with strictly positive entries, and $\widetilde{W}$ is the relative entropy projection of W' onto S then

$$\Delta(U,W') \geq \Delta(U,\widetilde{W}) + \Delta(\widetilde{W},W').$$

Furthermore, this holds with equality when S is affine, which is the case here since S is the set of matrices whose rows and columns each sum to 1. Rearranging and noting that $\Delta(A,B)$ is non-negative yields Corollary 3 of Herbster and Warmuth (2001), which is the inequality we need:

$$\Delta(U,W') - \Delta(U,\widetilde{W}) = \Delta(\widetilde{W},W') \geq 0.$$

Combining this with the inequality of Lemma 3 gives the critical per-trial invariant:

$$\Delta(U,W) - \Delta(U,\widetilde{W}) \geq (1-e^{-\eta})(W \bullet L) - \eta(U \bullet L). \tag{8}$$

We now introduce some notation and bound the expected total loss by summing the above inequality over a sequence of trials. When considering a sequence of trials, L^t is the loss matrix at trial t, W^{t-1} is PermELearn's weight matrix W at the start of trial t (so W^0 is the initial weight matrix) and W^t is the updated weight matrix $\widetilde{W}$ at the end of the trial.

Theorem 4 *For any learning rate $\eta > 0$, any doubly stochastic matrices U and initial W^0, and any sequence of T trials with loss matrices $L^t \in [0,1]^{n\times n}$ (for $1 \leq t \leq T$), the expected loss of PermELearn is bounded by:*

$$\sum_{t=1}^{T} W^{t-1} \bullet L^t \leq \frac{\Delta(U,W^0) - \Delta(U,W^T) + \eta \sum_{t=1}^{T} U \bullet L^t}{1-e^{-\eta}} \ .$$

Proof Applying (8) to trial t gives:

$$\Delta(U,W^{t-1}) - \Delta(U,W^t) \geq (1-e^{-\eta})(W^{t-1} \bullet L^t) - \eta(U \bullet L^t).$$

By summing the above over all T trials we get:

$$\Delta(U,W^0) - \Delta(U,W^T) \geq (1-e^{-\eta}) \sum_{t=1}^{T} W^{t-1} \bullet L^t - \eta \sum_{t=1}^{T} U \bullet L^t.$$

The bound then follows by solving for the total expected loss, $\sum_{t=1}^{T} W^{t-1} \bullet L^t$, of the algorithm. ■

When the entries of W^0 are all initialized to $\frac{1}{n}$ and U is a permutation then $\Delta(U,W^0) = n\ln n$. Since each doubly stochastic matrix U is a convex combination of permutation matrices, at least one minimizer of the total loss $\sum_{t=1}^{T} U \bullet L$ will be a permutation matrix. If $\mathcal{L}_{\text{best}}$ denotes the loss of such a permutation U^*, then Theorem 4 implies that the total loss of the algorithm is bounded by

$$\frac{\Delta(U^*,W^0) + \eta \mathcal{L}_{\text{best}}}{1-e^{-\eta}}.$$

If upper bounds $\Delta(U^*,W^0) \leq D_{\text{est}} \leq n\ln n$ and $\mathcal{L}_{\text{est}} \geq \mathcal{L}_{\text{best}}$ are known, then by choosing $\eta = \ln\left(1+\sqrt{\frac{2D_{\text{est}}}{\mathcal{L}_{\text{est}}}}\right)$, and the above bound becomes (Freund and Schapire, 1997):

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}} D_{\text{est}}} + \Delta(U^*,W^0). \tag{9}$$

A natural choice for D_{est} is $n\ln n$. In this case the tuned bound becomes

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}} n\ln n} + n\ln n.$$

5.3 Approximate Balancing

The preceding analysis assumes that PermELearn's weight matrix is perfectly balanced each iteration. However, balancing techniques are only capably of approximately balancing the weight matrix in finite time, so implementations of PermELearn must handle approximately balanced matrices. In Section 4.2, we describe an implementation that uses an approximately balanced $\widehat{W}^{t-1}$ at the start of iteration t rather than the completely balanced W^{t-1} of the preceding analysis. Lemma 1 shows that when this implementation of PermELearn uses an approximately balanced $\widehat{W}^{t-1}$ where each row and column sum is in $1 \pm \varepsilon_t$, then the expected loss on trial t is at most $W^{t-1} \bullet L^t + 3n^3\varepsilon_t$. Summing over all trials and using Theorem 4, this implementation's total loss is at most

$$\sum_{t=1}^{T} \left(W^{t-1} \bullet L^t + 3n^3\varepsilon_t \right) \leq \frac{\Delta(U, W^0) - \Delta(U, W^T) + \eta \sum_{t=1}^{T} U \bullet L^t}{1 - e^{-\eta}} + \sum_{t=1}^{T} 3n^3\varepsilon_t \ .$$

As discussed in Section 4.2, setting $\varepsilon_t = 1/(3n^3t^2)$ leads to an additional loss of less than $5/3$ over the bound of Theorem 4 and its subsequent tunings while incurring a total running time (over all T trials) in $O(Tn^6 \log(Tn))$. In fact, the additional loss for approximate balancing can be made less than any positive c by setting $\varepsilon_t = c/(5n^3t^2)$. Since the time to approximately balance depends only logarithmically on $1/\varepsilon$, the total time taken over T trials remains in $O(Tn^6 \log(Tn))$.

5.4 Split Analysis for the Hedge Algorithm

Perhaps the simplest case where the loss is linear in the parameter vector is the on-line allocation setting of Freund and Schapire (1997). It is instructive to apply our method of splitting the update in this simpler setting. There are N experts and the algorithm keeps a probability distribution w over the experts. In each trial the algorithm picks expert i with probability w_i and then gets a loss vector $\ell \in [0,1]^N$. Each expert i incurs loss ℓ_i and the algorithm's expected loss is $w \cdot \ell$. Finally w is updated to $\widetilde{w}$ for the next trial.

The Hedge algorithm (Freund and Schapire, 1997) updates its weight vector to $\tilde{w}_i = \frac{w_i e^{-\eta \ell_i}}{\sum_j w_j e^{-\eta \ell_j}}$. This update can be motivated by a tradeoff between the un-normalized relative entropy to the old weight vector and expected loss in the last trial (Kivinen and Warmuth, 1999):

$$\widetilde{w} := \operatorname*{argmin}_{\sum_i \widehat{w}_i = 1} \left(\Delta(\widehat{w}, w) + \eta \ \widehat{w} \cdot \ell \right).$$

For vectors, the relative entropy is simply $\Delta(\widehat{w}, w) := \sum_i \widehat{w}_i \ln \frac{\widehat{w}_i}{w_i} + w_i - \widehat{w}_i$. As in the permutation case, we can split this update (and motivation) into two steps: setting each $w'_i = w_i e^{-\eta \ell_i}$ then $\widetilde{w} = w' / \sum_i w'_i$. These are the solutions to:

$$w' := \operatorname*{argmin}_{\widehat{w}} \left(\Delta(\widehat{w}, w) + \eta \ \widehat{w} \cdot \ell \right) \ \text{ and } \ \widetilde{w} := \operatorname*{argmin}_{\sum_i \widehat{w}_i = 1} \Delta(\widehat{w}, w').$$

The following lower bound has been shown on the progress towards any probability vector u serving as a comparator:[4]

$$\begin{aligned}\Delta(u,w)-\Delta(u,\widetilde{w}) &= -\eta\, u\cdot\ell-\ln\sum_i w_i e^{-\eta\ell_i}\\ &\geq -\eta\, u\cdot\ell-\ln\sum_i w_i(1-(1-e^{-\eta})\ell_i)\\ &\geq -\eta\, u\cdot\ell+w\cdot\ell\,(1-e^{-\eta})\,, \end{aligned}\tag{10}$$

where the first inequality uses $e^{-\eta x}\leq 1-(1-e^{-\eta})x$, for any $x\in[0,1]$, and the second uses $-\ln(1-x)\geq x$, for $x\in[0,1]$. Surprisingly the same inequality already holds for the un-normalized update:[5]

$$\Delta(u,w)-\Delta(u,w')=-\eta\, u\cdot\ell+\sum_i w_i(1-e^{-\eta\ell_i})\geq w\cdot\ell\,(1-e^{-\eta})-\eta\, u\cdot\ell.$$

Since the normalization is a projection w.r.t. a Bregman divergence onto a linear constraint satisfied by the comparator u, $\Delta(u,w')-\Delta(u,\widetilde{w})\geq 0$ by the Generalized Pythagorean Theorem (Herbster and Warmuth, 2001). The total progress for both steps is again Inequality (10).

With the key Inequality (10) in hand, it is easy to introduce trial dependent notation and sum over trails (as done in the proof of Theorem 4, arriving at the familiar bound for Hedge (Freund and Schapire, 1997): For any $\eta>0$, any probability vectors w^0 and u, and any loss vectors $\ell^t\in[0,1]^n$,

$$\sum_{t=1}^{T} w^{t-1}\bullet\ell^t\leq\frac{\Delta(u,w^0)-\Delta(u,w^T)+\eta\sum_{t=1}^{T}u\bullet\ell^t}{1-e^{-\eta}}\quad.\tag{11}$$

Note that the r.h.s. is actually constant in the comparator u (Kivinen and Warmuth, 1999), that is, for all u,

$$\frac{\Delta(u,w^0)-\Delta(u,w^T)+\eta\sum_{t=1}^{T}u\bullet\ell^t}{1-e^{-\eta}}=\frac{-\ln\sum_i w_i^0 e^{-\eta\ell_i^{\leq T}}}{1-e^{\eta}}.$$

The r.h.s. of the above equality is often used as a potential in proving bounds for expert algorithms. We discuss this further in Appendix B.

5.5 When to Normalize?

Probably the most surprising aspect about the proof methodology is the flexibility about how and when to project onto the constraints. Instead of projecting a nonnegative matrix onto all $2n$ constraints at once (as in optimization problem (7)), we could mimic the Sinkhorn balancing algorithm by first projecting onto the row constraints and then the column constraints and alternating until convergence. The Generalized Pythagorean Theorem shows that projecting onto *any* convex constraint that is satisfied by the comparator class of doubly stochastic matrices brings the weight matrix closer to *every* doubly stochastic matrix.[6] Therefore our bound on $\sum_t W^{t-1}\bullet L^t$ (Theorem 4) holds if the exponential updates are interleaved with any sequence of projections to some subsets of the

4. This is essentially Lemma 5.2 of Littlestone and Warmuth (1994). The reformulation of this type of inequality with relative entropies goes back to Kivinen and Warmuth (1999)
5. Note that if the algorithm does not normalize the weights then w is no longer a distribution. When $\sum_i w_i<1$, the loss $w\cdot L$ amounts to incurring 0 loss with probability $1-\sum_i w_i$, and predicting as expert i with probability w_i.
6. There is a large body of work on finding a solution subject to constraints via iterated Bregman projections (see, e.g., Censor and Lent, 1981).

constraints. However, if the normalization constraints are not enforced then W is no longer a convex combination of permutations. Furthermore, the exponential update factors only decrease the entries of W and without any normalization all of the entries of W can get arbitrarily small. If this is allowed to happen then the "loss" $W \bullet L$ can approach 0 for any loss matrix, violating the spirit of the prediction model.

There is a direct argument that shows that the same final doubly stochastic matrix is reached if we interleave the exponential updates with projections to any of the constraints as long as all $2n$ constraints hold at the end. To see this we partition the class of matrices with positive entries into equivalence classes. Call two such matrices A and B *equivalent* if there are diagonal matrices R and C with positive diagonal entries such that $B = RAC$. Note that $[RAC]_{i,j} = R_{i,i}A_{i,j}C_{j,j}$ and therefore B is just a rescaled version of A. Projecting onto any row and/or column sum constraints amounts to pre- and/or post-multiplying the matrix by some positive diagonal matrices R and C. Therefore if matrices A and B are equivalent then the projection of A (or B) onto a set of row and/or column sum constraints results in another matrix equivalent to both A and B.

The importance of equivalent matrices is that they balance to the same doubly stochastic matrix.

Lemma 5 *For any two equivalent matrices A and RAC, where the entries of A and the diagonal entries of R and C are positive,*

$$\operatorname*{argmin}_{\substack{\forall i:\sum_j \widehat{A}_{i,j}=1 \\ \forall j:\sum_i \widehat{A}_{i,j}=1}} \Delta(\widehat{A},A) \quad = \quad \operatorname*{argmin}_{\substack{\forall i:\sum_j \widehat{A}_{i,j}=1 \\ \forall j:\sum_i \widehat{A}_{i,j}=1}} \Delta(\widehat{A},RAC).$$

Proof The strict convexity of the relative entropy implies that both problems have a unique matrix as their solution. We will now reason that the unique solutions for both problems are the same. By using a Lagrangian (as in the proof of Lemma 2) we see that the solution of the left optimization problem is a square matrix with $\dot{r}_i A_{i,j} \dot{c}_j$ in position i, j. Similarly the solution of the problem on the right has $\ddot{r}_i R_{i,i}A_{i,j}C_{j,j} \ddot{c}_j$ in position i, j. Here the factors $\dot{r}_i, \ddot{r}_i$ function as row normalizers and $\dot{c}_j, \ddot{c}_j$ as column normalizers. Given a solution matrix $\dot{r}_i, \dot{c}_j$ to the left problem, then $\dot{r}_i/R_{i,i}, \dot{c}_j/C_{j,j}$ is a solution of the right problem of the same value. Also if $\ddot{r}_i, \ddot{c}_j$ is a solution of right problem, then $\ddot{r}_i R_{i,i}, \ddot{c}_j C_{j,j}$ is a solution to the left problem of the same value.

This shows that both minimization problems have the same value and the matrix solutions for both problems are the same and unique (even though the normalization factors $\dot{r}_i, \dot{c}_j$ of say the left problem are not necessarily unique). Note that its crucial for the above argument that the diagonal entries of R, C are positive. ■

The analogous phenomenon is much simpler in the weighted majority case: Two non-negative vectors a and b are *equivalent* if $a = cb$, where c is any nonnegative scalar, and again each equivalence class has exactly one normalized weight vector.

PermELearn's intermediate matrix $W'_{i,j} := W_{i,j}e^{-\eta L_{i,j}}$ can be written $W \circ M$ where $\circ$ denotes the *Hadamard* (entry-wise) *Product* and $M_{i,j} = e^{-\eta L_{i,j}}$. Note that the Hadamard product commutes with matrix multiplication by diagonal matrices, if C is diagonal and $P = (A \circ B)C$ then $P_{i,j} = (A_{i,j}B_{i,j})C_{j,j} = (A_{i,j}C_{j,j})B_{i,j}$ so we also have $P = (AC) \circ B$. Similarly, $R(A \circ B) = (RA) \circ B$ when R is diagonal.

Hadamard products also preserve equivalence. For equivalent matrices A and $B = RAC$ (for diagonal R and C) the matrices $A \circ M$ and $B \circ M$ are equivalent (although they are not likely to be equivalent to A and B) since $B \circ M = (RAC) \circ M = R(A \circ M)C$.

This means that any two runs of PermELearn-like algorithms that have the same bag of loss matrices and equivalent initial matrices end with equivalent final matrices even if they project onto different subsets of the constraints at the end of the various trials.

In summary the proof method discussed so far uses a relative entropy as a measure of progress and relies on Bregman projections as its fundamental tool. In Appendix B we re-derive the bound for PermELearn using the value of the optimization problem (5) as a potential. This value is expressed using the dual optimization problem and intuitively the application of the Generalized Pythagorean Theorem now is replaced by plugging in a non-optimal choice for the dual variables. Both proof techniques are useful.

5.6 Learning Mappings

We have an algorithm that has small regret against the best permutation. Permutations are a subset of all mappings from $\{1,\ldots,n\}$ to $\{1,\ldots,n\}$. We continue using Π for a permutation and introduce Ψ to denote an arbitrary mapping from $\{1,\ldots,n\}$ to $\{1,\ldots,n\}$. Mappings differ from permutations in that the n dimensional vector $(\Psi(i))_{i=1}^n$ can have repeats, that is, $\Psi(i)$ might equal $\Psi(j)$ for $i \neq j$. Again we alternately represent a mapping Ψ as an $n \times n$ matrix where $\Psi_{i,j} = 1$ if $\Psi(i) = j$ and 0 otherwise. Note that such square[7] *mapping matrices* have the special property that they have exactly one 1 in each row. Again the loss is specified by a loss matrix L and the loss of mapping Ψ is $\Psi \bullet L$.

It is straightforward to design an algorithm *MapELearn* for learning mappings with exponential weights: Simply run n independent copies of the Hedge algorithm for each of the n rows of the received loss matrices. That is, the r'th copy of Hedge always receives the r'th row of the loss matrix L as its loss vector. Even though learning mappings is easy, it is nevertheless instructive to discuss the differences with PermELearn.

Note that MapELearn's combined weight matrix is now a convex combination of mappings, that is, a "singly" stochastic matrix with the constraint that each row sums to one. Again, after the exponential update (3), the constraints are typically not satisfied any more, but they can be easily reestablished by simply normalizing each row. The row normalization only needs to be done once in each trial: no iterative process is needed. Furthermore, no fancy decomposition algorithm is needed in MapELearn: for (singly) stochastic weight matrix W, the prediction $\Psi(i)$ is simply a random element chosen from the row distribution $W_{i,*}$. This sampling procedure produces a mapping Ψ such that $W = \mathbb{E}(\Psi)$ and thus $\mathbb{E}(\Psi \bullet L) = W \bullet L$ as needed.

We can use the same relative entropy between the single stochastic matrices, and the lower bound on the progress for the exponential update given in Lemma 3 still holds. Also our main bound (Theorem 4) is still true for MapELearn and we arrive at the same tuned bound for the total loss of MapELearn:

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}} D_{\text{est}}} + \Delta(U^*, W^0),$$

where $\mathcal{L}_{\text{best}}$, $\mathcal{L}_{\text{est}}$, and D_{est} are now the total loss of the best mapping, a known upper bound on $\mathcal{L}_{\text{best}}$, and an upper bound on $\Delta(U^*, W^0)$, respectively. Recall that $\mathcal{L}_{\text{est}}$ and D_{est} are needed to tune the η parameter.

7. In the case of mappings the restriction to square matrices is not essential.

Our algorithm PermElearn for permutations may be seen as the above algorithm for mappings while enforcing the column sum constraints in addition to the row constraints used in MapELearn. Since PermELearn's row balancing "messes up" the column sums and vice versa, an interactive procedure (i.e., Sinkhorn Balancing) is needed to create to a matrix in which each row *and* column sums to one. The enforcement of the additional column sum constraints results in a doubly stochastic matrix, an apparently necessary step to produce predictions that are permutations (and an expected prediction equal to the doubly stochastic weight matrix).

When it is known that the comparator is a permutation, then the algorithm always benefits from enforcing the additional column constraints. In general we should always make use of any constraints that the comparator is known to satisfy (see, e.g., Warmuth and Vishwanathan, 2005, for a discussion of this).

As discussed in Section 4.1, if A' is a Sinkhorn-balanced version of a non-negative matrix A, then

$$\text{for any permutations } \Pi_1 \text{ and } \Pi_2, \quad \frac{\prod_i A_{i,\Pi_1(i)}}{\prod_i A_{i,\Pi_2(i)}} = \frac{\prod_i A'_{i,\Pi_1(i)}}{\prod_i A'_{i,\Pi_2(i)}}. \tag{12}$$

An analogous invariant holds for mappings: If A' is a row-balanced version of a non-negative matrix A, then

$$\text{for any mappings } \Psi_1 \text{ and } \Psi_2, \quad \frac{\prod_i A_{i,\Psi_1(i)}}{\prod_i A_{i,\Psi_2(i)}} = \frac{\prod_i A'_{i,\Psi_1(i)}}{\prod_i A'_{i,\Psi_2(i)}}.$$

However it is important to note that column balancing does not preserve the above invariant for mappings. In fact, permutations are the subclass of mappings where invariant 12 holds.

There is another important difference between PermELearn and MapELearn. For MapELearn, the probability of predicting mapping Ψ with weight matrix W is always the product $\prod_i W_{i,\Psi(i)}$. The analogous property does *not* hold for PermELearn. Consider the balanced 2×2 weight matrix W on the right of Figure 2. This matrix decomposes into $\frac{\sqrt{2}}{1+\sqrt{2}}$ times the permutation $(1,2)$ plus $\frac{1}{1+\sqrt{2}}$ times the permutation $(2,1)$. Thus the probability of predicting with permutation $(1,2)$ is $\sqrt{2}$ times the probability of permutation $(2,1)$ for the PermELearn algorithm. However, when the probabilities are proportional to the intuitive product form $\prod_i W_{i,\Pi(i)}$, then the probability ratio for these two permutations is 2. Notice that this intuitive product weight measure is the distribution used by the Hedge algorithm that explicitly treats each permutation as a separate expert. Therefore PermELearn is clearly different than a concise implementation of Hedge for permutations.

6. Follow the Perturbed Leader Algorithm

Perhaps the simplest on-line algorithm is the *Follow the Leader* (FL) algorithm: at each trial predict with one of the best models on the data seen so far. Thus FL predicts at trial t with an expert in $\operatorname{argmin}_i \ell_i^{<t}$ or any permutation in $\operatorname{argmin}_\Pi \Pi \bullet L^{<t}$, where "$<t$" indicates that we sum over the past trials, that is, $\ell_i^{<t} := \sum_{q=1}^{t-1} \ell_i^q$. The FL algorithm is clearly non-optimal; in the expert setting there is a simple adversary strategy that forces FL to have loss at least n times larger than the loss of the best expert in hindsight.

The expected total loss of tuned Hedge is one times the loss of the best expert plus lower order terms. Hedge achieves this by randomly choosing experts. The probability w_i^{t-1} for choosing expert i at trial t is proportional to $e^{-\eta \ell_i^{<t}}$. As the learning rate $\eta \to \infty$, Hedge becomes FL (when there are

no ties) and the same holds for PermELearn. Thus the exponential weights with moderate η may be seen as a soft min calculation: the algorithm hedges its bets and does not put all its probability on the expert with minimum loss so far.

The "Follow the Perturbed Leader" (FPL) algorithm of Kalai and Vempala (2005) is an alternate on-line prediction algorithm that works in a very general setting. It adds random perturbations to the total losses of the experts incurred so far and then predicts with the expert of minimum perturbed loss. Their FPL* algorithm has bounds closely related to Hedge and other multiplicative weight algorithms and in some cases Hedge can be simulated exactly (Kuzmin and Warmuth, 2005) by judiciously choosing the distribution of perturbations. However, for the permutation problem the bounds we were able to obtain for FPL* are weaker than the the bound we obtained bounds for PermELearn that uses exponential weights despite the apparent similarity between our representations and the general formulation of FPL*.

The FPL setting uses an abstract k-dimensional decision space used to encode predictors as well as a k-dimensional state space used to represent the losses of the predictors. At any trial, the current loss of a particular predictor is the dot product between that predictor's representation in the decision space and the state-space vector for the trial. This general setting can explicitly represent each permutation and its loss when $k = n!$. The FPL setting also easily handles the encodings of permutations and losses used by PermELearn by representing each permutation matrix Π and loss matrix L as n^2-dimensional vectors.

The FPL* algorithm (Kalai and Vempala, 2005) takes a parameter ε and maintains a cumulative loss matrix C (initially C is the zero matrix) At each trial, FPL*:

1. Generates a random perturbation matrix P where each $P_{i,j}$ is proportional to $\pm r_{i,j}$ where $r_{i,j}$ is drawn from the standard exponential distribution.

2. Predicts with a permutation Π minimizing $\Pi \bullet (C+P)$.

3. After getting the loss matrix L, updates C to $C+L$.

Note that FPL* is more computationally efficient than PermELearn. It takes only $O(n^3)$ time to make its prediction (the time to compute a minimum weight bipartite matching) and only $O(n^2)$ time to update C. Unfortunately the generic FPL* loss bounds are not as good as the bounds on PermELearn. In particular, they show that the loss of FPL* on any sequence of trials is at most[8]

$$(1+\varepsilon)\mathcal{L}_{\text{best}} + \frac{8n^3(1+\ln n)}{\varepsilon}$$

where ε is a parameter of the algorithm. When the loss of the best expert is known ahead of time, ε can be tuned and the bound becomes

$$\mathcal{L}_{\text{best}} + 4\sqrt{2\mathcal{L}_{\text{best}} n^3(1+\ln n)} + 8n^3(1+\ln n) \quad .$$

Although FPL* gets the same $\mathcal{L}_{\text{best}}$ leading term, the excess loss over the best permutation grows as $n^3 \ln n$ rather the $n \ln n$ growth of PermELearn's bound. Of course, PermELearn pays for the improved bound by requiring more computation.

8. The n^3 terms in the bounds for FPL are n times the sum of the entries in the loss matrix. So if the application has a loss motif whose entries sum to only n, then the n^3 factors become n^2.

It is important to note that Kalai and Vempala also present a refined analysis of FPL* when the perturbed leader changes only rarely. This analysis leads to bounds that are similar to the bounds given by the entropic analysis of the Hedge algorithm (although the constant on the square-root term is not quite as good). However, this refined analysis cannot be directly applied with the efficient representations of permutations because the total perturbations associated with different permutations are no longer independent exponentials. We leave the adaptation of the refined analysis to the permutation case as an open problem.

7. Lower Bounds

In this section we prove lower bounds on the worst-case regret of any algorithm for our permutation learning problem by reducing the expert allocation problem for n experts with loss range $[0,n]$ to the permutation learning problem. We then show in Appendix C a lower bound for this n expert allocation problem that uses a known lower bound in the expert advice setting with losses in $[0,1]$.

For the reduction we choose any set of n permutations $\{\Pi^1,\ldots,\Pi^n\}$ that use disjoint positions, that is, $\sum_{i=1}^n \Pi^i$ is the $n\times n$ matrix of all ones. Using disjoint positions ensures that the losses of these n permutations can be set independently. Each Π^i matrix in this set corresponds to the ith expert in the n-expert allocation problem. To simulate an n-expert trial with loss vector $\ell\in[0,n]^n$ we use a loss matrix L s.t. $\Pi^i\bullet L=\ell_i$. This is done by setting all entries in $\{L_{q,\Pi^i(q)} : 1\le q\le n\}$ to $\ell_i/n\in[0,1]$, that is, $L=\sum_i \Pi^i(\ell_i/n)$. Now for any doubly stochastic matrix W,

$$W\bullet L=\sum_i \frac{\Pi^i\bullet W}{n}\,\ell_i.$$

Note that the n dimensional vector with the components $(\Pi^i\bullet W)/n$ is a probability vector and therefore any algorithm for the n-element permutation problem can be used as an algorithm for the n-expert allocation problem with losses in the range $[0,n]$. Thus any lower bound for the latter model is also a lower bound on the n-element permutation problem.

We first prove a lower bound for the case when at least one expert has loss zero for the entire sequence of trials. If the algorithm allocates any weight to experts that have already incurred positive loss, then the adversary can assign loss only to those experts and force the algorithm increase its expected loss without reducing the number of experts of loss zero. Thus we can assume w.l.o.g. that the algorithm allocates positive weight only to experts of zero loss. The algorithm minimizes its expected loss and the adversary maximizes it. We get a lower bound by fixing the adversary: This adversary assigns loss n to one of the experts which received the highest probability by the algorithm and all other experts are assigned loss zero. Clearly the optimal allocation against such an adversary uses the uniform distribution over those experts with zero loss. The number of experts with loss zero is reduced by one in each trial. At trial $t=1,\ldots,n-1$, $n+1-t$ experts are left and the expected loss is $\frac{n}{n+1-t}$. In the first $n-1$ trials the algorithm incurs expected loss

$$\sum_{i=2}^{n}\frac{n}{i}\approx n\ln n.$$

When the loss of the best expert is large then the following theorem follows from Corollary 11:

Theorem 6 *There exists n_0 such that for each dimension $n \geq n_0$, there is a T_n where for any number of trials $T \geq T_n$ the following holds for any algorithm A for learning permutations of n elements in the allocation setting: there is a sequence S of T trials such that*

$$\mathcal{L}_{\text{best}}(S) \leq nT/2 \quad \textit{and} \quad \mathcal{L}_A(S) - \mathcal{L}_{\text{best}}(S) \geq \sqrt{(nT/2)\, n \ln n}.$$

These two lower bounds can be combined to the following lower bound on the expected regret for our permutation learning problem:

$$\max\left(\sqrt{\mathcal{L}_{\text{best}} n \ln n}, n \ln n\right) \geq \frac{\sqrt{\mathcal{L}_{\text{best}} n \ln n} + n \ln n}{2}.$$

This means that the tuned upper bound on the expected regret of PermELearn given after Theorem 4 cannot be improved by more than a small ($2\sqrt{2}$) constant factor.

8. Conclusions

We considered the problem of learning a permutation on-line, when the per-trial loss is specified by a matrix $L \in [0,1]^{n \times n}$ and the loss of a permutation matrix Π is the linear loss $\Pi \bullet L$. The standard approach would treat each permutation as an expert. However this is computationally inefficient and introduces an additional factor of n in the regret bounds (since the per-trial loss of a permutation is $[0,n]$ rather than $[0,1]$). We do not know if this factor of n is necessary for permutations, and it remains open whether their special structure allows better regret bounds on the standard expert algorithms when the experts are permutations.

We developed a new algorithm called PermELearn that uses a doubly stochastic matrix to maintain its uncertainty over the hidden permutation. PermELearn decomposes this doubly stochastic matrix into a small mixture of permutation matrices and predicts with a random permutation from this mixture. A similar decomposition was used by Warmuth and Kuzmin (2008) to learn as well as the best fixed-size subset of experts.

PermELearn belongs to the Exponentiated Gradient family of updates and the analysis uses a relative entropy as a measure of progress. The main technical insight is that the per-trial progress bound already holds for the un-normalized update and that re-balancing the matrix only increases the progress. Since the re-balancing step does not have a closed form, accounting for it in the analysis would otherwise be problematic. We also showed that the update for the Hedge algorithm can be split into an un-normalized update and a normalization. In this more basic setting the per trial progress bound also holds for the un-normalized update.

Our analysis techniques rely on Bregman projection methods[9] and the regret bounds hold not only for permutations but also for mixtures of permutations. This means that if we have additional convex constraints that are satisfied by the mixture that we compare against, then we can project the algorithm's weight matrix onto these constraints without hurting the analysis (Herbster and Warmuth, 2001). With these kinds of side constraints we can enforce some relationships between the parameters, such as $W_{i,j} \geq W_{i,k}$ (i is more likely mapped to j than k).

Our main contribution is showing how to apply the analysis techniques from the expert advice setting to the problem of efficiently learning a permutation. This means that many of the tools from

9. Following Kuzmin and Warmuth (2007), we also showed in Appendix B that the regret bounds proven in this paper can be reproduced with potential based methods.

the expert setting are likely to carry over to permutations: lower bounding the weights when the comparator is shifting (Herbster and Warmuth, 1998), long-term memory when shifting between a small set of comparators (Bousquet and Warmuth, 2002), capping the weights from the top if the goal is to be close to the best set of disjoint permutations of fixed size (Warmuth and Kuzmin, 2008), adapting the updates to the multi-armed bandit setting when less feedback is provided (Auer et al., 2002),[10] and PAC Bayes analysis of the exponential updates (McAllester, 2003).

We also applied the "Follow the Perturbed Leader" techniques to our permutation problem. This algorithm adds randomness to the total losses and then predicts with a minimum weighted matching which costs $O(n^3)$ whereas our more complicated algorithm is at least $O(n^4)$ and has precision issues. However the bounds currently provable for the FPL* algorithm of Kalai and Vempala (2005) are much worse than for our PermELearn algorithm. The key open problem is whether we can have the best of both worlds: add randomness to the loss matrix so that the expected minimum weighted matching is the stochastic matrix produced by the PermELearn update (4). This would mean that we could use the faster algorithm together with our tighter analysis. In the simpler weighted majority setting this has been done already (Kuzmin and Warmuth, 2005; Kalai, 2005). However we do not yet know how to simulate the PermELearn update this way.

Our on-line learning problem requires that the learner's prediction to be an actual permutation. This requirement makes sense for the linear loss we focus on in this paper, but may be less appropriate for on-line regression problems. Consider the case where on each trial the algorithm selects a doubly stochastic matrix M while nature simultaneously picks a matrix $X \in [0,1]^{n\times n}$ and a real number y. The prediction is $\hat{y} = M \bullet X$ and the loss on the trial is $(\hat{y}-y)^2$. With this convex quadratic loss, it is generally better for the algorithm to hedge its bets between competing permutations and select its doubly stochastic parameter matrix W as M instead of a random permutation matrix Π chosen s.t. $\mathbb{E}(\Pi) = W$. The Exponentiated Gradient algorithm can be applied to this type of non-linear regression problem (see, e.g., Helmbold et al., 1999) and Sinkhorn Balancing can project the parameter matrix W onto the row and column sum constraints.

We close with an open problem involving higher order loss functions. In this paper we considered linear losses specified by a square matrix L where $L_{i,j}$ gives the loss when entry (i, j) is used in the permutation. Can one prove good regret bounds when the loss depends on how the permutation assigns multiple elements? A pairwise loss could be represented with a four-dimensional matrix L where $L_{i,j,k,l}$ is added to the loss only when the predicted permutation maps *both* i to j *and* k to l. The recently developed Fourier analysis techniques for permutations (Kondor et al., 2007; Huang et al., 2009) may be helpful in generalizing our techniques to this kind of higher order loss.

Acknowledgments

We thank Vishy Vishwanathan for helping us simplify the lower bounds, and David DesJardins for helpful discussions and pointers to the literature on Sinkhorn Balancing.

10. Recently, a less efficient algorithm which explicitly maintains one expert per permutation has been analyzed in the bandit setting by Cesa-Bianchi and Lugosi (2009). However the bounds they obtain have the loss range as an additional factor in the regret bound (a factor of n for permutations).

Appendix A. Size of the Decomposition

Here we show that the iterative matching method of Algorithm 1 requires most n^2-2n+2 permutations to decompose an doubly stochastic matrix. This matches the bound provided by Birkhoff's Theorem. Note that the discussion in Section 4 shows why Algorithm 1 can always find a suitable permutation.

Theorem 7 *Algorithm 1 decomposes any doubly stochastic matrix into a convex combination of at most n^2-2n+2 permutations.*

Proof Let W be a doubly stochastic matrix and let $\Pi_1,\ldots,\Pi_\ell$ and $\alpha_1,\ldots,\alpha_\ell$ be any sequence of permutations and coefficients created by Algorithm 1 on input W. For $0\le j\le\ell$, define $M^j = W-\sum_{i=1}^{j}\alpha_i\Pi_i$. By permuting rows and columns we can assume without loss of generality that Π_ℓ is the identity permutation. Let G^j (for $1\le j\le\ell$) be the (undirected) graph on the n vertices $\{1,\ldots n\}$ where the undirected edge $\{p,q\}$ between nodes $p\neq q$ is present if and only if either $M^j_{p,q}$ or $M^j_{q,p}$ is non-zero. Thus both $G^{\ell-1}$ and G^ℓ are the empty graph and each G^{j+1} has a (not necessarily strict) subset of the edges in G^j. Note the natural correspondences between vertices in the graphs and rows and columns in the matrices.

The proof is based in the following key invariant:

$$\#\text{ of zero entries in } M^j \ge j + (\#\text{ connected components in } G^j) - 1.$$

This holds for the initial M^0. Furthermore, when the connected components of G^j and G^{j+1} are the same, the algorithm insures that M^{j+1} has at least one more zero than M^j. We now analyze the case when new connected components are created.

Let vertex set V be a connected component in G^{j+1} that was split off a larger connected component in G^j. We overload the notation, and use V also for the set of matrix rows and/or columns associated with the vertices in the connected component.

Since V is a connected component of G^{j+1} there are no edges going between V and the rest of the graph, so if M^{j+1} is viewed as a (conserved) flow, there is no flow either into or out of V:

$$\sum_{r\in V}\sum_{c\notin V} M^{j+1}_{r,c} = \sum_{r\notin V}\sum_{c\in V} M^{j+1}_{r,c} = 0.$$

Thus all entries of M^j in the sets $\{M^j_{r,c}>0 : r\in V, c\notin V\}$ and $\{M^j_{r,c}>0 : r\notin V, c\in V\}$ are set to zero in M^{j+1}. Since V was part of a larger connected component in G^j, at least one of these sets must be non-empty. We now show that both these sets of entries are non-empty.

Each row and column of M^j sum to $1-\sum_{i=1}^{j}\alpha_i$. Therefore

$$\left(1-\sum_{i=1}^{j}\alpha_i\right)|V| = \sum_{r\in V}\sum_{c=1}^{n} M^j_{r,c} = \sum_{c\in V}\sum_{r=1}^{n} M^j_{r,c}.$$

By splitting the inner sums we get:

$$\sum_{r\in V}\sum_{c\in V} M^j_{r,c} + \sum_{r\in V}\sum_{c\notin V} M^j_{r,c} = \sum_{c\in V}\sum_{r\in V} M^j_{r,c} + \sum_{c\in V}\sum_{r\notin V} M^j_{r,c}.$$

By canceling the first sums and viewing M^j as a flow in G^j we conclude that the total flow out of V in M^j equals the total flow into V in M^j, that is,

$$\sum_{r\in V}\sum_{c\notin V} M^j_{r,c} = \sum_{c\in V}\sum_{r\notin V} M^j_{r,c}$$

and both sets $\{M^j_{r,c} > 0 : r \in V, c \notin V\}$ and $\{M^j_{r,c} > 0 : r \notin V, c \in V\}$ sum to the same positive total, and thus are non-empty.

This establishes the following fact that we can use in the remainder of the proof: for each new connected component V in G^{j+1}, some entry $M^j_{r,c}$ from a row r in V was set to zero.

Now let k_j (and k_{j+1}) be the number of connected components in graph G^j (and G^{j+1} respectively). Since the edges in G^{j+1} are a subset of the edges in G^j, $k_{j+1} \geq k_j$. We already verified the invariant when $k_j = k_{j+1}$, so we proceed assuming $k_{j+1} > k_j$. In this case at most $k_j - 1$ components of G^j survive when going to G^{j+1}, and at least $k_{j+1} - (k_j - 1)$ new connected components are created. The vertex sets of the new connected components are disjoint, and in the rows corresponding to each new connected component there is at least one non-zero entry in M^j that is zero in M^{j+1}. Therefore, M^{j+1} has at least $k_{j+1} - k_j + 1$ more zeros than M^j, verifying the invariant for the case when $k_{j+1} > k_j$.

Since $G^{\ell-1}$ has n connected components, the invariant shows that the number of zeros in $M^{\ell-1}$ is at least $\ell - 1 + n - 1$. Furthermore, M^ℓ has n more zeros than $M^{\ell-1}$, so M^ℓ has at least $\ell + 2n - 2$ zeros. Since M^ℓ has only n^2 entries, $n^2 \geq \ell + 2n - 2$ and $\ell \leq n^2 - 2n + 2$ as desired. ■

The fact that Algorithm 1 uses at most $n^2 - 2n + 2$ permutations can also be established with a dimensionality argument like that in Section 2.7 of Bazaraa et al. (1977).

Appendix B. Potential Based Bounds

Let us begin with the on-line allocation problem in the simpler expert setting. There are always two ways to motivate on-line updates. One trades the divergence to the last weight vector against the loss in the last trial, and the other trades the divergence to the initial weight vector against the loss in all past trials (Azoury and Warmuth, 2001):

$$w^t := \operatorname*{argmin}_{\sum_i w_i = 1} \left(\Delta(w, w^{t-1}) + \eta\, w \cdot \ell^t\right), \quad w^t := \operatorname*{argmin}_{\sum_i w_i = 1} \left(\Delta(w, w^0) + \eta\, w \cdot \ell^{\leq t}\right).$$

By differentiating the Lagrangian for each optimization problem we obtain the solutions to both minimization problems:

$$w^t_i = w^{t-1}_i e^{-\eta \ell^t_i + \widetilde{\beta}^t}, \quad w^t_i = w^0_i e^{-\eta \ell^{\leq t}_i + \beta^t},$$

where the signs of the Lagrange multipliers $\widetilde{\beta}^t$ and β^t are unconstrained and their values are chosen so that the equality constraints are satisfied. The left update can be unrolled to obtain

$$w^t_i = w^0_i e^{-\eta \ell^{\leq t}_i + \sum_{q=1}^t \widetilde{\beta}^q}.$$

This means the Lagrangian multipliers for both problems are related by the equality $\sum_{q=1}^t \widetilde{\beta}^q = \beta^t$ and both problems have the same solution:[11] $w^t_i = \frac{w^0_i e^{-\eta \ell^{\leq t}_i}}{\sum_{j=1}^n w^0_j e^{-\eta \ell^{\leq t}_j}}$. We use the value of the right convex

11. The solutions can differ if the minimization is over linear inequality constraints (Kuzmin and Warmuth, 2007).

optimization problem's objective function as our potential v^t. Its Lagrangian is

$$\sum_i \left(w_i \ln \frac{w_i}{w_i^0} + w_i^0 - w_i + \eta w_i \ell_i^{\leq t} \right) + \beta \left(\sum_i w_i - 1 \right)$$

and since there is no duality gap:[12]

$$v^t := \min_{\sum_i w_i = 1} \left(\Delta(w, w^0) + \eta \, w \cdot \ell_{\leq t} \right) = \max_{\beta} \underbrace{\sum_i w_i^0 (1 - e^{-\eta \ell_i^{\leq t} - \beta}) - \beta}_{\text{dual function } \theta^t(\beta)}.$$

Here β is the (unconstrained) dual variable for the primal equality constraint and the w_i's have been optimized out. By differentiating we can optimize β in the dual problem and arrive at

$$v^t = -\ln \sum_i w_i^0 e^{-\eta \ell_i^{\leq t}}.$$

This form of the potential has been used extensively for analyzing expert algorithms (see, e.g., Kivinen and Warmuth, 1999; Cesa-Bianchi and Lugosi, 2006). One can easily show the following key inequality (essentially Lemma 5.2 of Littlestone and Warmuth, 1994):

$$\begin{aligned} v^t - v^{t-1} &= -\ln \sum_i w_i^0 e^{-\eta \ell_i^{\leq t}} + \ln \sum_i w_i^0 e^{-\eta \ell_i^{\leq t}} \\ &= -\ln \sum_i w_i^{t-1} e^{-\eta \ell_i^t} \\ &\geq -\ln \sum_i w_i^{t-1} (1 - (1 - e^{-\eta}) \ell_i^t) \\ &\geq (1 - e^{-\eta}) \, w^{t-1} \cdot \ell^t. \end{aligned} \tag{13}$$

Summing over all trials and using $v^0 = 0$ gives the familiar bound:

$$\sum_{t=1}^{T} w^{t-1} \cdot \ell^t \leq \frac{v^T}{1 - e^{-\eta}} = \frac{1}{1 - e^{-\eta}} \min_{\sum_i w_i = 1} \left(\Delta(w, w^0) + \eta \, w \cdot \ell^{\leq T} \right).$$

Note that by Kivinen and Warmuth (1999)

$$v^t - v^{t-1} = -\ln \left(\sum_i w_i^{t-1} e^{-\eta \ell_i^t} \right) = \Delta(u, w^{t-1}) - \Delta(u, w^t) + \eta \, u \cdot \ell^t,$$

and therefore the Inequality (13) is the same as Inequality (10). Since

$$\sum_{t=1}^{T} (v^t - v^{t-1}) = v^T = -\ln \left(\sum_i w_i^0 e^{-\eta \ell_i^{\leq t}} \right),$$

summing the bound (13) over t coincides with the bound (11).

12. There is no duality gap in this case because the primal problem is a feasible convex optimization problem subject to linear constraints.

We now reprove the key inequality (13) using the dual function $\theta^t(\beta)$. Note β^t maximizes this function, that is, $v^t = \theta^t(\beta^t)$, and the optimal primal solution is $w_i^t = w_i^0 e^{-\eta \ell_i^{\leq t} - \beta^t}$. Now,

$$\begin{aligned} v^t - v^{t-1} &= \theta^t(\beta^t) - \theta^{t-1}(\beta^{t-1}) \\ &\geq \theta^t(\beta^{t-1}) - \theta^{t-1}(\beta^{t-1}) \\ &= \sum_i \underbrace{w_i^0 e^{-\eta \ell_i^{<t} - \beta^{t-1}}}_{w_i^{t-1}} (1 - e^{-\eta \ell_i^t}) \\ &\geq \sum_i w_i^{t-1} (1 - (1 - (1 - e^{-\eta})\ell_i^t)) \\ &= (1 - e^{-\eta})\, w^{t-1} \cdot \ell_t \end{aligned}$$

where we used $e^{-\eta \ell_i^t} \leq 1 - (1 - e^{-\eta})\ell_i^t$ to get the fourth line. Notice that in the first inequality above we used $\theta^t(\beta^t) \geq \theta^t(\beta^{t-1})$. This is true because β^t maximizes $\theta^t(\beta)$ and the old choice β^{t-1} is non-optimal. The dual parameter β^{t-1} assures that w^{t-1} is normalized and $\theta^t(\beta^{t-1})$ is related to plugging the intermediate unnormalized weights $w'^t_i := w_i^0 e^{-\eta \ell_i^{\leq t} - \beta^{t-1}}$ into the primal problem for trial t. This means that the inequality $\theta^t(\beta^t) \geq \theta^t(\beta^{t-1})$ corresponds to the Bregman projection of the unnormalized update onto the equality constraint. The difference $\theta^t(\beta^{t-1}) - \theta^{t-1}(\beta^{t-1})$ in the second line above is the progress in the value when going from w^{t-1} at the end of trial $t-1$ to the intermediate unnormalized update w'^t at trial t. Therefore this proof also does not exploit the normalization.

The bound for the permutation problem follows the same outline. We use the value of the following optimization problem as our potential:

$$\begin{aligned} v_{t+1} &:= \min_{\substack{\forall i : \sum_j A_{i,j} = 1 \\ \forall j : \sum_i A_{i,j} = 1}} \left(\Delta(A, W^0) + \eta\, (A \bullet L^{\leq t})\right) \\ &= \max_{\alpha_i, \beta_j} \underbrace{\sum_{i,j} W_{i,j}^0 (1 - e^{-\eta L_{i,j}^{\leq t} - \alpha_i - \beta_j}) - \sum_i \alpha_i - \sum_j \beta_j}_{\theta^t(\alpha, \beta)}. \end{aligned}$$

The α_i and β_j are the dual variables for the row and column constraints. Now we can't optimize out the dual variables in the dual function $\theta^t(\alpha, \beta)$ does not have a maximum in closed form. Nevertheless the above proof technique based on duality still works. Let α^t and β^t be the optimizers of $\theta^t(\alpha, \beta)$. Then the optimum primal solution (the parameter vector of PermELearn) becomes

$$W_{i,j}^t = W_{i,j}^0 e^{-\eta L_{i,j}^{\leq t} - \alpha_i^t - \beta_j^t}$$

and we can analyze the increase in value as before:

$$\begin{aligned} v^t - v^{t-1} &= \theta^t(\alpha^t, \beta^t) - \theta^{t-1}(\alpha^{t-1}, \beta^{t-1}) \\ &\geq \theta^t(\alpha_{t-1}, \beta_{t-1}) - \theta^{t-1}(\alpha^{t-1}, \beta^{t-1}) \\ &= \sum_{i,j} \underbrace{W_{i,j}^0 e^{-\eta L_{i,j}^{<t} - \alpha_i^{t-1} - \beta_j^{t-1}}}_{W_{i,j}^{t-1}} (1 - e^{-\eta L_{i,j}^{t-1}}) \\ &\geq \sum_{i,j} W_{i,j}^{t-1} (1 - (1 - e^{-\eta}) L_{i,j}^t) \\ &= (1 - e^{-\eta})\, W^{t-1} \bullet L^t. \end{aligned}$$

Summing over trials in the usual way gives the bound

$$\sum_{t=1}^{T} W^{t-1} \bullet L^t \leq \frac{v_T}{1-e^{-\eta}} = \frac{1}{1-e^{-\eta}} \min_{\substack{\forall i: \sum_j A_{i,j}=1 \\ \forall j: \sum_i A_{i,j}=1}} \left(\Delta(A, W^0) + \eta\, (A \bullet L_{\leq T})\right)$$

which is the same as the bound of Theorem 4.

Appendix C. Lower Bounds for the Expert Advice Setting

We first modify a known lower bound from the expert advice setting with the absolute loss (Cesa-Bianchi et al., 1997). We begin by describing that setting and show how it relates to the allocation setting for experts.

In the expert advice setting there are n experts. Each trial t starts with nature selecting a *prediction* x_i^t in [0,1] for each expert $i \in \{1,\ldots,n\}$. The algorithm is given these predictions and then produces its own prediction $\hat{y}^t \in [0,1]$. Finally, nature selects a *label* $y^t \in \{0,1\}$ for the trial. The algorithm is charged loss $|\hat{y}^t - y^t|$ and expert i gets loss $|x_i^t - y^t|$.

Any algorithm in the allocation setting leads to an algorithm in the above expert advice setting: keep the weight update unchanged, predict with the weighted average (i.e., $\hat{y}^t = w^{t-1} \cdot \mathbf{x}^t$) and define the loss vector in $\ell^t \in [0,1]^n$ in terms of the absolute loss:

$$|\underbrace{w^{t-1} \cdot \mathbf{x}^t}_{\hat{y}^t} - y^t| \;=\; \sum_i w_i^{t-1} \underbrace{|x_i^t - y^t|}_{\ell_i^t} \;=\; w^{t-1} \cdot \ell^t,$$

where the first equality holds because $x_i^t \in [0,1]$ and $y^t \in \{0,1\}$. This means that any lower bound on the regret in the above expert advice setting immediately leads to a lower bound on the expected loss in the allocation setting for experts when the loss vectors lie in $[0,1]^n$.

We now introduce some more notation and state the lower bound from the expert advice setting that we build on. Let $\mathcal{S}_{n,T}$ be the set of all sequences of T trials with n experts in the expert advice setting with the absolute loss. Let $V_{n,T}$ be the minimum over algorithms of the worst case regret over sequences in $\mathcal{S}_{n,T}$.

Theorem 8 *(Cesa-Bianchi et al., 1997, Theorem 4.5.2)*

$$\lim_{n\to\infty} \lim_{T\to\infty} \frac{V_{n,T}}{\sqrt{(T/2)\ln n}} = 1.$$

This means that for all $\varepsilon > 0$ there exists n_ε such that for each $n \geq n_\varepsilon$, there is a $T_{\varepsilon,n}$ where for all $T \geq T_{\varepsilon,n}$,

$$V_{n,T} \geq (1-\varepsilon)\sqrt{(T/2)\ln n}.$$

By further expanding the definition of $V_{n,T}$ we get the following version of the above lower bound that avoids the use of limits:

Corollary 9 *For all $\varepsilon > 0$ there exists n_ε such that for each number of experts $n \geq n_\varepsilon$, there is a $T_{\varepsilon,n}$ where for any number of trials $T \geq T_{\varepsilon,n}$ the following holds for any algorithm A in the expert advice setting with the absolute loss: there is a sequence S of T trials with n experts such that*

$$\mathcal{L}_A(S) - \mathcal{L}_{\text{best}}(S) \geq (1-\varepsilon)\sqrt{(T/2)\ln n}.$$

This lower bound on the regret depends on the number of trials T. We now use a reduction to bound $\mathcal{L}_{\text{best}}(S)$ by $T/2$. Define $R(S^-)$ as the transformation that takes a sequence S^- of trials in $\mathcal{S}_{n-1,T}$ and produces a sequence of trials in $\mathcal{S}_{n,T}$ by adding an extra expert whose predictions are simply 1 minus the predictions of the first expert. On each trial the absolute loss of the additional expert on sequence $R(S^-)$ is 1 minus the loss of the first expert. Therefore either the first expert or the additional expert will have loss at most $T/2$ on $R(S^-)$.

Theorem 10 *For all $\varepsilon > 0$ there exists n_ε such that for each number of experts $n \geq n_\varepsilon$, there is a $T_{\varepsilon,n}$ where for any number of trials $T \geq T_{\varepsilon,n}$ the following holds for any algorithm A in the expert advice setting with the absolute loss: there is a sequence S of T trials with n experts such that*

$$\mathcal{L}_{\text{best}}(S) \leq T/2 \quad \textit{and} \quad \mathcal{L}_A(S) - \mathcal{L}_{\text{best}}(S) \geq (1-\varepsilon)\sqrt{(T/2)\ln n}.$$

Proof We begin by showing that the regret on a transformed sequence in $\{R(S^-) : S^- \in \mathcal{S}_{n-1,T}\}$ is at least $(1-\varepsilon/2)\sqrt{(T/2)\ln(n-1)}$.

Note that for all $R(S^-)$, $\mathcal{L}_{\text{best}}(R(S^-)) \leq T/2$ and assume to the contrary that some algorithm A has regret strictly less than $(1-\varepsilon/2)\sqrt{(T/2)\ln(n-1)}$ on every sequence in $\{R(S^-) : S^- \in \mathcal{S}_{n-1,T}\}$. We then create an algorithm A^- that runs transformation $R(\cdot)$ on-the-fly and predicts as A does on the transformed sequence. Therefore A^- on S^- and A on $R(S^-)$ make the same predictions and have the same total loss. On every sequence $S^- \in \mathcal{S}_{n-1,T}$ we have $\mathcal{L}_{\text{best}}(S^-) \geq \mathcal{L}_{\text{best}}(R(S))$ and therefore

$$\begin{aligned}
\mathcal{L}_{A^-}(S^-) - \mathcal{L}_{\text{best}}(S^-) &\leq \mathcal{L}_{A^-}(S^-) - \mathcal{L}_{\text{best}}(R(S^-)) \\
&= \mathcal{L}_A(R(S^-)) - \mathcal{L}_{\text{best}}(R(S^-)) \\
&< (1-\varepsilon/2)\sqrt{(T/2)\ln(n-1)}.
\end{aligned}$$

Now if $n-1$ is at least the $n_{\varepsilon/2}$ of Corollary 9 and T is at least the $T_{\varepsilon/2,n-1}$ of the same corollary, then this contradicts that corollary.

This means that for any algorithm A and large enough n and T, there is a sequence S for which the algorithm has regret at least $(1-\varepsilon/2)\sqrt{(T/2)\ln(n-1)}$ and $\mathcal{L}_{\text{best}}(S) \leq T/2$. By choosing the lower bound on n large enough,

$$(1-\varepsilon/2)\sqrt{(T/2)\ln(n-1)} \geq (1-\varepsilon)\sqrt{(T/2)\ln n}$$

and the theorem follows. ■

Note that the tuned upper bounds in the allocation setting (9) have an additional factor of $\sqrt{2}$. This is due to the fact that in the allocation setting the algorithm predicts with the weighted average and this is non-optimal. In the expert setting with the absolute loss, the upper bound (based on a different prediction function) and the lower bound on the regret are asymptotically tight (See Theorem 8). We are now ready to prove our lower bound for the allocation setting with experts when the losses of the experts are in $[0,n]^n$ instead of $[0,1]^n$.

Corollary 11 *There exists n_0 such that for each dimension $n \geq n_0$, there is a T_n where for any number of trials $T \geq T_n$ the following holds for any algorithm A for allocation setting with n experts: there is a sequence S of T trials with loss vectors in $[0,n]^n$ such that*

$$\mathcal{L}_{\text{best}}(S) \leq nT/2 \quad \textit{and} \quad \mathcal{L}_A(S) - \mathcal{L}_{\text{best}}(S) \geq \sqrt{(nT/2)n\ln n}.$$

Proof Via the reduction we stated at the beginning of the section, the following lower bound for the allocation setting with n experts immediately follows from the previous theorem: For any algorithm in the allocation setting for n experts there is a sequence $\widetilde{S}$ of T trials where the losses of the experts lie in $[0,1]$ such that

$$\mathcal{L}_{\text{best}}(\widetilde{S}) \leq T/2 \quad \text{and} \quad \mathcal{L}_A(\widetilde{S}) - \mathcal{L}_{\text{best}}(\widetilde{S}) \geq \sqrt{(T/2)\ln n}.$$

Now we simply scale the loss vectors by the factor n, that is, the scaled sequences S have loss vectors in the range $[0,n]^n$ and $\mathcal{L}_{\text{best}}(S) \leq nT/2$. The lower bound becomes $n\sqrt{(T/2)\ln n} = \sqrt{(nT/2)n\ln n}$. ■

References

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

K. Azoury and M. K. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Journal of Machine Learning*, 43(3):211–246, June 2001. Special issue on *Theoretical Advances in On-line Learning, Game Theory and Boosting*, edited by Yoram Singer.

H. Balakrishnan, I. Hwang, and C. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *43rd IEEE Conference on Decision and Control*, pages 4874–4879, December 2004.

M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. Wiley, second edition, 1977.

R. Bhatia. *Matrix Analysis*. Springer-Verlag, Berlin, 1997.

A. Blum, S. Chawla, and A. Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36:249–260, 2003.

O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Physics*, 7:200–217, 1967.

Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34(3):321–353, July 1981.

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

N. Cesa-Bianchi and G. Lugosi. Combinatorial bandits. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT 09)*, 2009.

N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, May 1997.

Y. Chen, L. Fortnow, N. Lambert, D. Pennock, and J. Wortman. Complexity of combinatorial market makers. In *Ninth ACM Conference on Electronic Commerce (EC '08)*. ACM Press, July 2008.

J. Franklin and J. Lorenz. On the scaling of multidimensional matrices. *Linear Algebra and its applications*, 114/115:717–735, 1989.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

M. Fürer. Quadratic convergence for scaling of matrices. In *Proceedings of ALENEX/ANALCO*, pages 216–223. SIAM, 2004.

Geoffrey J. Gordon. No-regret algorithms for online convex programs. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 489–496. MIT Press, 2006.

D. P. Helmbold and R. E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(01):51–68, 1997.

D. P. Helmbold, J. Kivinen, and M. K. Warmuth. Relative loss bounds for single neurons. *IEEE Transactions on Neural Networks*, 10(6):1291–1304, November 1999.

M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998. Earlier version in 12th ICML, 1995.

M. Herbster and M. K. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.

J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *Journal of Machine Learning Research*, 10:997–1070, 2009.

M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, July 2004.

A. Kalai. Simulating weighted majority with FPL. Private communication, 2005.

A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005. Special issue Learning Theory 2003.

B. Kalantari and L. Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its applications*, 240:87–103, 1996.

J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Information and Computation*, 132(1):1–64, January 1997.

J. Kivinen and M. K. Warmuth. Averaging expert predictions. In *Computational Learning Theory, 4th European Conference (EuroCOLT '99), Nordkirchen, Germany, March 29-31, 1999, Proceedings*, volume 1572 of *Lecture Notes in Artificial Intelligence*, pages 153–167. Springer, 1999.

R. Kondor, A. Howard, and T. Jebara. Multi-object tracking with representations of the symmetric group. In *Proc. of the 11th International Conference on Artificial Intelligence and Statistics*, March 2007.

D. Kuzmin and M. K. Warmuth. Optimum follow the leader algorithm. In *Proceedings of the 18th Annual Conference on Learning Theory (COLT '05)*, pages 684–686. Springer-Verlag, June 2005. Open problem.

D. Kuzmin and M. K. Warmuth. Online Kernel PCA with entropic matrix updates. In *Proceedings of the 24rd international conference on Machine learning (ICML '07)*, pages 465–471. ACM International Conference Proceedings Series, June 2007.

N. Linial, A. Samorodnitsky, and A. Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inform. Comput.*, 108(2): 212–261, 1994. Preliminary version in FOCS '89.

D. McAllester. PAC-Bayesian stochastic model selection. *Machine Learning*, 51(1):5–21, 2003.

R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Staticstics*, 35(2):876–879, June 1964.

E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4:773–818, 2003.

V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.

M. K. Warmuth and D. Kuzmin. Randomized PCA algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9:2217–2250, 2008.

M. K. Warmuth and S.V.N. Vishwanathan. Leaving the span. In *Proceedings of the 18th Annual Conference on Learning Theory (COLT '05)*, Bertinoro, Italy, June 2005. Springer-Verlag. Journal version in progress.