# Online Learning of Permutations Using Extended Formulation

**Holakou Rahmanian, David P. Helmbold, and S.V.N. Vishwanathan**
Department of Computer Science, University of California Santa Cruz
{holakou, dph, vishy}@ucsc.edu

## Abstract

We introduce a new method for efficiently learning permutations that exploits the technique of *extended formulation* from the combinatorial optimization community. This *extended formulation* technique encodes the hard-to-describe polytope of permutations as the projection of an easier to describe polytope in a higher dimensional space. Although the best special-purpose algorithm for permutations has a slightly better regret bound, our methodology yields regret bounds like those of other published algorithms. The new methodology can be generalized to other combinatorial objects. It also has an elegant method of determining the algorithm's initial weight vector and the use of the extended formulation leads to a faster and more natural way to make appropriate predictions.

## 1 Introduction

This paper introduces an efficient and effective online learning algorithm over permutations of a set of elements for scheduling problems. The introduced learning algorithm follows a methodology which can be generalized to other combinatorial objects. Online learning algorithms are being successfully applied to an increasing variety of problems, so it is important to have good tools and techniques for creating good algorithms that match the particular problem at hand.

The online learning setting proceeds in a series of trials where the algorithm makes a prediction with a permutation and then receives the loss of its choice. The algorithm can then update its internal representation based on this feedback and the process moves on to the next trial. Unlike batch learning settings, there is no assumed distribution from which losses are randomly drawn, instead the losses are drawn adversarially. In general, an adversary can force arbitrarily large loss on the algorithm, so instead of measuring the algorithm's performance by the total loss incurred, the algorithm is measured by its *regret*, the amount of loss the algorithm incurs above that of the single best permutation in hindsight.

Concretely, in each trial $t$, the algorithm would predict a permutation of $n$ *jobs* to be processed, and then obtain a loss vector in $\boldsymbol{\ell}^t \in [0,1]^n$ which indicates the *processing time* of each job. The loss of the algorithm on that trial is the *sum of completion times* which is the inner product of the given loss vector and the predicted permutation in first order representation in $n$-dimensional space[1]. Over a series of trials, the regret of the algorithm is the difference in combined sum of completion time between the online algorithm and the single best permutation chosen in hindsight. Therefore the regret of the algorithm can be viewed as the cost of not knowing the best permutation ahead of time.

One way to create algorithms for these combinatorial problems (e.g. learning permutations) is to use one of the well-known so-called "experts algorithms" like weighted majority [15] or hedge [7] with

[1] As in [20], the loss family provided in our approach is linear over the first order representation of the objects [6]. Concretely, for permutations of $n$ items, we work with vectors $\boldsymbol{v} \in \mathbb{R}^n$ in which each of the elements of $\{1, 2, \ldots, n\}$ appears exactly once. In contrast, [9] works with the second order representation, and consequently losses, which is a more general loss family (See [20] for comparison).

each combinatorial object treated as an "expert". However, this requires explicitly keeping track of one weight for each of the exponentially many combinatorial objects, and thus results in an inefficient algorithm. There has been much work on creating efficient algorithms that implicitly encode the weights over the set of combinatorial objects using a concise representations (e.g. for permutations, see [1, 9, 20]).

The component hedge algorithm of Koolen, Warmuth, and Kivinen [14] is a powerful generic technique when the implicit encodings are suitably simple. It works by performing multiplicative updates on the parameters of its implicit representation. However, the implicit representation is typically constrained to lie in a convex polytope. Therefore Bregman projections are used after the update to return the implicit representation to the desired polytope. Note that this process is only efficient when there are simply a small (polynomial) number of constraints on the implicit representations.

The combinatorial optimization community have studied the problem of concisely specifying the convex hulls of complicated combinatorial structures like permutations using few constraints. They have developed a powerful technique of representing these polytopes as a linear projection of a higher-dimensional polyhedron using *extended formulations* so that the polytope description has way less (polynomial instead of exponential) constraints. There have been several works aimed at efficiently describing the polytope of different combinatorial objects like permutations [8] and Huffman trees [16]. Discovered by the combinatorial optimization community, extended formulation, on which our results rely, is a general methodology to nicely describe combinatorial polyhedra [11, 12].

**Contributions:** The main contributions of this paper are:

1. The introduction of extended formulation techniques to the machine learning community. Our methodology uses a redundant representation introduced by extended formulation for the combinatorial objects where one part of the representation allows for a natural loss measure while another enables the simple specification of the class using only polynomially many constraints. To better match the extended formulations to the machine learning applications, we augment the extended formulation with slack variables.
2. A new and faster prediction technique. In most applications, the algorithm usually predicts by first re-expressing the algorithm's weight or usage vector as a small convex combination of combinatorial objects, and then randomly samples from the convex combination [9, 14, 20]. The redundant representation allows for a more direct and efficient way to generate the algorithm's random prediction, bypassing the need to create convex combinations.
3. A new and elegant initialization method. Component Hedge style loss bounds depend on the distance from the initial hypothesis to the best predictor in the class, and a roughly uniform initialization is usually a good choice. Rather than directly picking a feasible initialization, we introduce the idea of first creating an infeasible encoding with good distance properties, and then projecting it into the feasible polytope.

**Paper Outline:** The remainder of the paper is organized as follows. Section 2 explains the extended formulation used in our setting by reviewing the work by Kaibel and Pashkovich [12]. We then propose our algorithm along its technical steps in Section 3. Finally, Section 4 concludes with contrasting our approach to existing ones and discussing the generality of our method[2].

## 2 Extended Formulation

There are many combinatorial objects whose polytope can only be described using exponentially many facets in its original space (e.g. see [16]). In order to have more efficient algorithms, there have been several efforts in the field of combinatorial optimization towards describing these polytopes in some other spaces. In recent years, the concept of representing these polytopes as a linear projection of a higher-dimensional polyhedron – which is known as *extended formulation* – has received significant attention. There are many combinatorial objects whose associated polyhedra can be described as

[2]Due to lack of space, we have omitted the proofs and detailed discussions. For the longer and more general version of the paper please see the preprint in [17].

the projection of a higher dimensional polytope with far fewer facets back into the original space. See [11] for some of the tools for constructing such extended formulations. In this section, we first overview the work by Kaibel and Pashkovich [12], and then adapt the formulation to fit our methodology.

**Constructing Extended Formulation from Reflection Relations:** One of the tools to construct polynomial size extended formulations is the framework developed by [12] using *reflection relations*. The basic idea is to start with a corner of the polytope (e.g. a permutation) and then create a sequence of reflections through hyperplanes (e.g. swapping a pair of elements) so that any corner of the polytope can be generated by applying a subsequence of the reflections to the canonical corner. The convex hull is then generated by allowing "partial reflections" (i.e. containing the entire line segment connecting the original point and its reflection). Any point in the convex hull is then created by a sequence of partial reflections, and can be encoded by a sequence of variables indicating how much of each reflection was used. In fact, there is no need to start with a single corner, one could consider passing an entire polytope as an input through the sequence of (partial) reflections to generate a new polytope.

This framework provides an inductive construction of higher dimensional polytopes via a sequence of reflection relations (see Theorem 1 in [12]). Assume we have an extended formulation for $n$-element permutations. We create an extended formulation for the $(n+1)$-element permutations by first mapping the $n$-element permutations to a subset of the $(n+1)$-element permutations. Then an appropriate sequence of reflection relations is used to generate the rest of the $(n+1)$-element permutations. Adding the variables and constraints for this sequence of reflection relations to the extended formulation for the $n$-element permutations yields an extended formulation for the $(n+1)$-element permutations. Theorem 1 in [12] provides the sufficient conditions for the correctness of this procedure.

By doing this inductive step, there will be additional variables and inequalities in the formulations. In fact, for each reflection relation, there will be one additional variable indicating the extent to which the reflection occurs, and two additional inequalities indicating the two extreme cases of complete reflection and remaining unchanged. Therefore, if polynomially many reflection relations are used to go from $n$ to $n+1$, then we can construct an extended formulation of polynomial size for the convex hull of $n$-element permutations for all $n$.

**Extended Formulation of Objects Closed under Re-Ordering:** Assume we want to construct an extended formulation for the class of combinatorial objects which is closed under any re-ordering. Note that permutations belong to such class of objects. In this case, for reflection relations, one can use hyperplanes going through the origin with normal vector of form $\boldsymbol{e}_i - \boldsymbol{e}_j$ which makes reflections as swapping $i$th and $j$th elements. Now let us figure out the form of extended formulation in this particular case. First, we find the additional variable along with two additional inequalities associated with this reflection relation. Concretely, assume $\boldsymbol{v} \in \mathbb{R}^n$ is going through this reflection relation and $\boldsymbol{v}' \in \mathbb{R}^n$ is the output. Since $\boldsymbol{v}'$ must fall into the line segment connecting $\boldsymbol{v}$ and its reflection via the hyperplane, we have $\exists x \in \mathbb{R}\ \boldsymbol{v}' - \boldsymbol{v} = x\,(\boldsymbol{e}_i - \boldsymbol{e}_j)$ and $\langle \boldsymbol{e}_i - \boldsymbol{e}_j, \boldsymbol{v} \rangle \leq \langle \boldsymbol{e}_i - \boldsymbol{e}_j, \boldsymbol{v}' \rangle \leq -\langle \boldsymbol{e}_i - \boldsymbol{e}_j, \boldsymbol{v} \rangle$. Using these two properties, one can obtain the relation between $\boldsymbol{v}'$ and $\boldsymbol{v}$ via a linear transformation given the additional variable $x$ as well as the constraints enforced on $x$:

$$\boldsymbol{v}' = \boldsymbol{m}\, x + \boldsymbol{v} \quad \text{where } \boldsymbol{m} = \boldsymbol{e}_i - \boldsymbol{e}_j, \qquad 0 \leq x \leq v_j - v_i.^{3} \tag{1}$$

Notice that $x$ indicates the value that is being swapped between $i$th and $j$th elements which can go from zero (remaining unchanged) to the maximum swap capacity (complete swap). Also note that the sequence of reflection relations can be viewed as a sequence of comparators in a network [2]. In the case of permutations, the comparators associated with combined sequences of reflection relations form bubble-sort networks as in each inductive step a sequence of length $n$ of corresponding comparators must bubble-out the largest element. Furthermore, since the construction of the polytope is inductive, the order of reflection relations is the opposite of the direction of the network of comparators. In fact, one can observe that the extended formulation of a permutation is the swap values of the comparators in the sorting network. Moreover, a mixture of permutations can be represented by partial swap values in the comparators (See Figure 1).

[3] $v_j - v_i$ may already be in terms of previous variables in extended formulation.

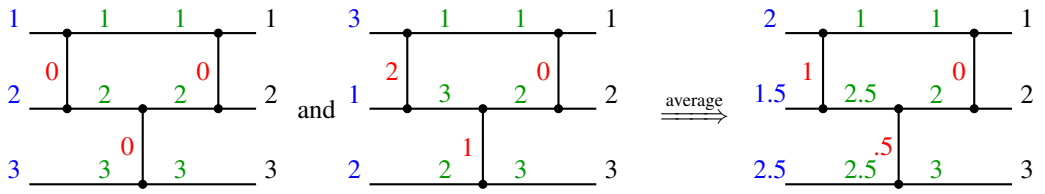


Figure 1: An example of sorting network illustrating the extended formulation in pure and mixture permutations of $n = 3$ items. The values of $\boldsymbol{v}$ and $\boldsymbol{x}$ are shown in blue and red, respectively.

Now we can express the extended formulation using the results above. Suppose we are using $m$ reflection relations in total. Then starting from an *anchor point* $\boldsymbol{c}$ and applying the equation in (1) successively, we obtain the affine transformation connecting the extended formulation space $\mathcal{X}$ and original space $\mathcal{V}$:

$$\boldsymbol{v} = M\,\boldsymbol{x} + \boldsymbol{c}, \quad \boldsymbol{v}, \boldsymbol{c} \in \mathcal{V} \subset \mathbb{R}^n,\ \boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^m,\ M \in \{-1, 0, 1\}^{n \times m}.$$

The anchor point $\boldsymbol{c}$ is the identity permutation $[1, 2, \ldots, n]^T$. Also using all the inequalities from the reflection relations introduced in (1), the extended formulation space $\mathcal{X}$ can be described as $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^m : A\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{x} \geq \boldsymbol{0}\}$ in which $A \in \mathbb{R}^{m \times m}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

***Note:*** In description above, for simplicity, we explained the basic inductive approach of building extended formulation for permutations. However, one can build an extended formulation for permutations non-inductively and more efficiently based on an arbitrary sorting network.

**Combinatorial Polytope Description in Augmented Formulation:** Even though the polytope can now be described using polynomial number of facets in extended formulation $\boldsymbol{x} \in \mathcal{X}$, it is not natural to define linear loss over the elements of $\boldsymbol{x}$. However, one can concatenate the original formulation $\boldsymbol{v} \in \mathcal{V}$ to the extended formulation $\boldsymbol{x} \in \mathcal{X}$, so that it is possible to not only efficiently describe the polytope, but also provide the meaningful loss of *sum of completion times* for permutations. In addition, in order to have the comfort of working with affine subspaces[4], we add a positive slack vector $\boldsymbol{\lambda}$ to turn all inequalities into equalities. As a result, we define the *augmented formulation space* $\mathcal{W}$:

$$\mathcal{W} = \{(\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{\lambda}) \in \mathbb{R}_+^{n+2m} \mid A\boldsymbol{x} + \boldsymbol{\lambda} = \boldsymbol{b} \text{ and } \boldsymbol{v} = M\boldsymbol{x} + \boldsymbol{c}\}$$

## 3 Algorithm

In this section, we propose our algorithm XF-Perm, discuss its technical steps, and prove its regret bounds. The main idea of our algorithm is to maintain a mean vector over all instances of the objects by keeping track of an evolving point $\boldsymbol{w}^t = (\boldsymbol{v}^t, \boldsymbol{x}^t, \boldsymbol{\lambda}^t)$ in the augmented formulation space $\mathcal{W}$ through trials $t = 1, \ldots, T$. Since $\boldsymbol{v}$ is the only constituent over which the loss vector $\boldsymbol{\ell}^t$ is defined, we work with $\boldsymbol{L}^t = (\boldsymbol{\ell}^t, \boldsymbol{0}, \boldsymbol{0}) \in [0, 1]^{n+2m}$ in the augmented formulation space $\mathcal{W}$. The main structure of XF-Perm is shown in Algorithm 1. Similar to [14], our algorithm consists of three main technical parts: *Prediction*, *Update*, and *Projection*.

**Prediction:** Predict with an instance $\boldsymbol{\gamma}^{t-1} \in \mathcal{H}$ such that $\mathbb{E}\left[\boldsymbol{\gamma}^{t-1}\right] = \boldsymbol{v}^{t-1}$.

We describe how to probabilisticly predict with an instance such that it has the same expected value as the mixture vector of which we are keeping track. In order to achieve efficient prediction, despite the vastly common idea in the literature [9, 14, 19, 20], one can avoid decomposition and do prediction directly.

To this purpose, we introduce a distribution $\mathcal{D}$ with efficient sampling such that $\mathbb{E}_{\mathcal{D}}[\boldsymbol{\gamma}] = \boldsymbol{v}$. We define $x_i/(x_i + \lambda_i)$ as *swap probability* associated with the $i$th comparator $i \in \{1..m\}$. Algorithm 2 incorporates the notion of swap probabilities to construct an efficient sampling procedure from a distribution $\mathcal{D}$ which has the right expectation.

[4] The reader will see that having affine subspaces allows us to take an efficient projection approach, namely iterative Bregman projections.

**Algorithm 1** XF-Perm

$\boldsymbol{w}^0 = (\boldsymbol{v}^0, \boldsymbol{x}^0, \boldsymbol{s}^0) \in \mathcal{W}$
For $t = 1, \ldots, T$
  Set $\boldsymbol{\gamma}^{t-1} \leftarrow$ **Prediction**$(\boldsymbol{w}^{t-1})$ where $\boldsymbol{\gamma}^{t-1} \in \mathcal{H}$ is a random object s.t. $\mathbb{E}\left[\boldsymbol{\gamma}^{t-1}\right] = \boldsymbol{v}^{t-1}$
  Incur a loss $\boldsymbol{\gamma}^{t-1} \cdot \boldsymbol{\ell}^t$
  **Update**:
    Set $\widehat{\boldsymbol{v}}_i^{t-1} \leftarrow v_i^{t-1}\, e^{-\eta\, \ell_i^t}$ for all $i \in \{1..n\}$
  Set $\boldsymbol{w}^t \leftarrow$ **Projection**$\underbrace{(\widehat{\boldsymbol{v}}^{t-1}, \boldsymbol{x}^{t-1}, \boldsymbol{\lambda}^{t-1})}_{\widehat{\boldsymbol{w}}^{t-1}}$ where $\boldsymbol{w}^t = \underset{\boldsymbol{w} \in \mathcal{W}}{\arg\min}\ \Delta\left(\boldsymbol{w} || \widehat{\boldsymbol{w}}^{t-1}\right)$

**Lemma 1.** *(i) Given $(\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{\lambda}) \in \mathcal{W}$, the Algorithm 2 samples $\boldsymbol{\gamma} \in \mathcal{H}$ from a $\mathcal{D}$ such that $\mathbb{E}_{\mathcal{D}}[\boldsymbol{\gamma}] = \boldsymbol{v}$. (ii) The time complexity of Algorithm 2 is $O(m)$.*

**Algorithm 2** Fast-Prediction

**Input:** $(\boldsymbol{x}, \boldsymbol{\lambda}) \in \mathbb{R}_+^{2m}$
**Output:** A prediction $\boldsymbol{\gamma} \in \mathcal{H}$
$\boldsymbol{\gamma} \leftarrow \boldsymbol{c}$
**for** $k = 1$ to $m$ **do**
  $(i_k, j_k) \leftarrow$ wire indices associated with the $k$-th comparator
  **if** $x_k = 0$ **then**
    **continue**
  **else**
    Switch the $i_k$th and $j_k$th components of $\boldsymbol{\gamma}$ w.p. $x_k/(x_k + \lambda_k)$.
  **end if**
**end for**
**return** $\boldsymbol{\gamma}$

**Update:** Update the mixture $\boldsymbol{w}^{t-1}$ to $\widehat{\boldsymbol{w}}^{t-1}$ according to the incurred loss multiplicatively.
Having defined $\boldsymbol{L}^t = (\boldsymbol{\ell}^t, \boldsymbol{0}, \boldsymbol{0})$, the updated $\widehat{\boldsymbol{w}}^{t-1}$ is obtained using a trade-off between the linear loss and the unnormalized relative entropy [14]:

$$\widehat{\boldsymbol{w}}^{t-1} = \underset{\boldsymbol{w} \in \mathbb{R}^r}{\arg\min}\ \Delta(\boldsymbol{w} || \boldsymbol{w}^{t-1}) + \eta\, \boldsymbol{w} \cdot \boldsymbol{L}^t, \quad \text{where} \quad \Delta(\boldsymbol{a} || \boldsymbol{b}) = \sum_i a_i \log \frac{a_i}{b_i} + b_i - a_i.$$

Using Lagrange multipliers, it is fairly straight-forward to see that only the $\boldsymbol{v}$ components of $\boldsymbol{w}^{t-1}$ are updated:

$$\forall i \in \{1..n\},\ \widehat{v}_i^{t-1} = v_i^{t-1}\, e^{-\eta\, \ell_i^t}; \qquad \widehat{\boldsymbol{x}}^{t-1} = \boldsymbol{x}^{t-1}; \qquad \widehat{\boldsymbol{\lambda}}^{t-1} = \boldsymbol{\lambda}.$$

**Projection:** Project the updated mixture $\widehat{\boldsymbol{w}}^{t-1}$ back to the polytope $\mathcal{W}$ and obtain $\boldsymbol{w}^t$.
We use an unnormalized relative entropy Bregman projection to project $\widehat{\boldsymbol{w}}^{t-1}$ back into $\mathcal{W}$ obtaining the new $\boldsymbol{w}^t$ for the next trial.

$$\boldsymbol{w}^t = \underset{\boldsymbol{w} \in \mathcal{W}}{\arg\min}\ \Delta(\boldsymbol{w} || \widehat{\boldsymbol{w}}^{t-1}) \tag{2}$$

Let $\Psi_0, \ldots, \Psi_{n+m-1}$ be the $n + m$ hyperplanes where the $n$ constraints of $\boldsymbol{v} = M\boldsymbol{x} + \boldsymbol{c}$ and the $m$ constraints of $A\boldsymbol{x} + \boldsymbol{\lambda} = \boldsymbol{b}$ are satisfied.Then $\mathcal{W}$ is the intersection of the $\Psi_k$'s. Since the non-negativity constraints are already enforced by the definition of $\Delta(\cdot||\cdot)$, it is possible to solve (2) using iterative Bregman projections[5] [4]. Starting from $\boldsymbol{p}_0 = \widehat{\boldsymbol{w}}^{t-1}$, we iteratively compute:

$$\boldsymbol{p}_k = \underset{\boldsymbol{p} \in \Psi_{(k \bmod m+n)}}{\arg\min}\ \Delta(\boldsymbol{p} || \boldsymbol{p}_{k-1})$$

repeatedly cycling through the constraints. [5] discusses how one can efficiently project onto each hyperplane $\Psi_k$ for all $k \in \{1..n+m\}$. It is known that $\boldsymbol{p}_k$ converges in norm to the unique solution of (2) [3, 4].

[5] In [9] Sinkhorn balancing is used for projection which is also a special case of iterative Bregman projection.

| Algorithm | Regret Bound |
|---|---|
| OnlineRank [1] | $O(n^2\sqrt{T})$ |
| **XF-Perm** | $O(n^2(\log n)^{\frac{1}{2}}\sqrt{T})$ |
| PermELearn [9] | $O(n^2(\log n)^{\frac{1}{2}}\sqrt{T})$ |
| PermutahedLearn [20] | $O(n^2(\log n)^{\frac{1}{2}}\sqrt{T})$ |
| Follow the Perturbed Leader [13] | $O(n^{\frac{5}{2}}\sqrt{T})$ |
| Hedge Algorithm [7] | $O(n^{\frac{5}{2}}(\log n)^{\frac{1}{2}}\sqrt{T})$ |

Table 1: Comparing the regret bounds of XF-Perm with other existing algorithms.

**Regret Bounds:** Similar to [14], the general regret bound depends on the initial weight vector $\boldsymbol{w}^0 \in \mathcal{W}$ via $\Delta(\boldsymbol{w}(\boldsymbol{\gamma})||\boldsymbol{w}^0)$ where $\boldsymbol{w}(\boldsymbol{\gamma}) \in \mathcal{W}$ is the augmented formulation of the permutation $\boldsymbol{\gamma} \in \mathcal{H}$ against which the algorithm is compared (the best $\boldsymbol{\gamma}$ for the adversarially chosen sequence of losses).

**Lemma 2.** *Let $L^* := \min_{\boldsymbol{\gamma}\in\mathcal{H}} \sum_{t=1}^{T} \boldsymbol{\gamma}\cdot\boldsymbol{\ell}^t$. By proper tuning of the learning rate $\eta$:*

$$\mathbb{E}\left[\sum_{t=1}^{T} \boldsymbol{\gamma}^{t-1}\cdot\boldsymbol{\ell}^t\right] - \min_{\boldsymbol{\gamma}\in\mathcal{H}} \sum_{t=1}^{T} \boldsymbol{\gamma}\cdot\boldsymbol{\ell}^t \le \sqrt{2L^*\,\Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0)} + \Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0)$$

The proof uses standard techniques from the online learning literature (see, e.g. , [14]). In order to get good bounds, the initial weight $\boldsymbol{w}^0$ must be "close" to all corners $\boldsymbol{\gamma}$ of the polytope, and thus in the"middle" of $\mathcal{W}$. In previous works [9, 14, 20], the initial weight is explicitly chosen and it is often set to be the uniform usage of the permutations.

Here, instead of explicitly selecting $\boldsymbol{w}^0 \in \mathcal{W}$, we implicitly design the initial point. First, we find an intermediate "middle" point $\widetilde{\boldsymbol{w}}^0 \in \mathbb{R}^{n+2m}$ with good distance properties, and then project $\widetilde{\boldsymbol{w}}^0$ into $\mathcal{W}$ to obtain the initial $\boldsymbol{w}^0$ for the first trial. A good choice for $\widetilde{\boldsymbol{w}}^0$ is $n\,\mathbf{1}$ where $\mathbf{1} \in \mathbb{R}^{n+2m}$ is the vector of all ones. This leads to the nice bound $\Delta(\boldsymbol{w}(\boldsymbol{h})||\widetilde{\boldsymbol{w}}^0) \le 3mn$ for all permutations $\boldsymbol{\gamma} \in \mathcal{H}$. The Generalized Pythagorean Theorem [10] ensures that the same bound holds for $\boldsymbol{w}^0$. This leads to the guarantee below:

**Theorem 3.** *If each $\boldsymbol{\ell}^t \in [0,1]^n$ and the weights are initialized to $\boldsymbol{w}^0 = \arg\min_{\boldsymbol{w}\in\mathcal{W}} \Delta(\boldsymbol{w}||n\,\mathbf{1})$, then XF-Perm's regret is:*

$$\mathbb{E}\left[\sum_{t=1}^{T} \boldsymbol{\gamma}^{t-1}\cdot\boldsymbol{\ell}^t\right] - \min_{\boldsymbol{\gamma}\in\mathcal{H}} \sum_{t=1}^{T} \boldsymbol{h}\cdot\boldsymbol{\ell}^t \le \sqrt{6L^*\,mn} + 3mn$$

*Using a sorting networks of size $m = \mathcal{O}(n\log n)$ [2], the regret will be $O(n^2(\log n)^{\frac{1}{2}}\sqrt{T})$.*

## 4 Comparisons and Conclusion

Using extended formulation, we introduce a new algorithm for learning permutations which can be also generalized to other combinatorial objects. Table 1 contains a comparison of the regret bounds for the new XF-Perm algorithm and the previous algorithms for permutations.

Our algorithm for permutations with reflection relations as extended formulation can be easily extended to learning Huffman trees for compressing $n$ symbols. We assume Huffman trees are represented by sequence of depths of the $n$ symbols in the tree, and the loss vector indicates the frequencies of the symbol – which leads to the *average code length* as loss of algorithm per trial [17]. Using additional comparators in the sorting network along with some linear maps, one can construct extended formulation for Huffman trees [12]. As a result, one can similarly derive an effective and efficient for online learning of Huffman trees.

In general, going beyond reflection relations, extended formulations can be used for encoding ill-behaved polytopes, and consequently, result in effective and efficient learning algorithms. [17] explores a general methodology of using extended formulation for online learning of combinatorial objects. As another concrete example, [18] uses the extended formulation induced by dynamic programming to develop learning algorithms for combinatorial objects such as binary search trees.

## References


[1] Nir Ailon. Improved bounds for online learning over the permutahedron and other ranking polytopes. In *AISTATS*, pages 29–37, 2014.

[2] Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting inc logn parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[3] Heinz H Bauschke and Jonathan M Borwein. Legendre functions and the method of random bregman projections. *Journal of Convex Analysis*, 4(1):27–67, 1997.

[4] Lev 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 mathematical physics*, 7(3):200–217, 1967.

[5] Inderjit S Dhillon and Joel A Tropp. Matrix nearness problems with bregman divergences. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1120–1146, 2007.

[6] Persi Diaconis. Group representations in probability and statistics. *Lecture Notes-Monograph Series*, 11:i–192, 1988.

[7] Yoav Freund and Robert 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, 1997.

[8] Michel X Goemans. Smallest compact formulation for the permutahedron. *Mathematical Programming*, 153(1):5–11, 2015.

[9] David P Helmbold and Manfred K Warmuth. Learning permutations with exponential weights. *The Journal of Machine Learning Research*, 10:1705–1736, 2009.

[10] Mark Herbster and Manfred K Warmuth. Tracking the best linear predictor. *The Journal of Machine Learning Research*, 1:281–309, 2001.

[11] Volker Kaibel. Extended formulations in combinatorial optimization. *arXiv preprint arXiv:1104.1023*, 2011.

[12] Volker Kaibel and Kanstantsin Pashkovich. Constructing extended formulations from reflection relations. In *Facets of Combinatorial Optimization*, pages 77–100. Springer, 2013.

[13] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[14] Wouter M Koolen, Manfred K Warmuth, and Jyrki Kivinen. Hedging structured concepts. 2010.

[15] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[16] Jean-François Maurras, Thanh Hai Nguyen, and Viet Hung Nguyen. On the convex hull of huffman trees. *Electronic Notes in Discrete Mathematics*, 36:1009–1016, 2010.

[17] Holakou Rahmanian, David P Helmbold, and SVN Vishwanathan. Online learning of combinatorial objects via extended formulation. *arXiv preprint arXiv:1609.05374*, 2017 (Submitted to ALT 2018).

[18] Holakou Rahmanian and Manfred K Warmuth. Online dynamic programming. In *Advances in Neural Information Processing Systems*, 2017.

[19] Manfred K Warmuth and Dima Kuzmin. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(10):2287–2320, 2008.

[20] Shota Yasutake, Kohei Hatano, Shuji Kijima, Eiji Takimoto, and Masayuki Takeda. Online linear optimization over permutations. In *Algorithms and Computation*, pages 534–543. Springer, 2011.