Online Learning of Combinatorial Objects via Extended Formulation

Holakou Rahmanian HOLAKOU@UCSC.EDU
David P. Helmbold DPH@UCSC.EDU
S.V.N. Vishwanathan VISHY@UCSC.EDU
Department of Computer Science
University of California Santa Cruz
Santa Cruz, CA 95060, USA

Editor: No editors

Abstract

The standard techniques for online learning of combinatorial objects perform multiplicative updates followed by projections into the convex hull of all the objects. However, this methodology can be expensive if the convex hull contains many facets. For example, the convex hull of n-symbol Huffman trees is known to have exponentially many facets (Maurras et al., 2010). We get around this difficulty by exploiting extended formulations (Kaibel, 2011), which encode the polytope of combinatorial objects in a higher dimensional "extended" space with only polynomially many facets. We develop a general framework for converting extended formulations into efficient online algorithms with good relative loss bounds. We present applications of our framework to online learning of Huffman trees and permutations. The regret bounds of the resulting algorithms are within a factor of $\mathcal{O}(\sqrt{\log(n)})$ of the state-of-the-art specialized algorithms for permutations, and depending on the loss regimes, improve on or match the state-of-the-art for Huffman trees. Our method is general and can be applied to other combinatorial objects.

Keywords: online learning, extended formulation, combinatorial object, hedge

1. Introduction

This paper introduces a general methodology for developing efficient and effective algorithms for learning combinatorial structures. Examples include learning the best permutation of a set of elements for scheduling or assignment problems, or learning the best Huffman tree for compressing sequences of symbols. 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.

Prediction Game 1 Prediction game for the combinatorial class $\mathcal{H} \subset \mathbb{R}_+^n$.

1: For each trial $t = 1, \ldots, T$
2: The **learner** predicts (perhaps randomly) with an object $\widehat{\boldsymbol{h}}^{t-1}$ in class $\mathcal{H}$.
3: The **adversary** reveals a loss vector $\boldsymbol{\ell}^t \in [0,1]^n$.
4: The **learner** incurs a (expected) linear loss $\mathbb{E}[\widehat{\boldsymbol{h}}^{t-1} \cdot \boldsymbol{\ell}^t]$.

©2017 Holakou Rahmanian, David P. Helmbold, and S.V.N. Vishwanathan.

The online learning setting proceeds in a series of trials where the algorithm makes a prediction or takes an action associated with an object in the appropriate combinatorial space and then receives the loss of its choice in such a way that the loss of any of the possible combinatorial objects can be easily computed (See Prediction Game 1). 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 predictor in some comparator class. Usually the comparator class is the class of objects in the combinatorial space being learned. To make the setting concrete, consider the case of learning Huffman trees for compression[1]. In each trial, the algorithm would (perhaps randomly) predict a Huffman tree, and then obtain a sequence of symbols to be encoded. The loss of the algorithm on that trial is the average bits per symbol to encode the sequence using the predicted Huffman tree. More generally, the loss could be defined as the inner product of any loss vector from the unit cube and the code lengths of the symbols. The total loss of the algorithm is the expected average bits per symbol summed over trials. The regret of the algorithm is the difference between its total loss and the sum over trials of the average bits per symbol for the single best Huffman tree chosen in hindsight. Therefore the regret of the algorithm can be viewed as the cost of not knowing the best combinatorial object ahead of time. With proper tuning, the regret is typically logarithmic in the number of combinatorial objects.

One way to create algorithms for these combinatorial problems is to use one of the well-known so-called "experts algorithms" like Randomized Weighted Majority (Littlestone and Warmuth, 1994) or Hedge (Freund and Schapire, 1997) with each combinatorial object is 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. Furthermore, it also causes an additional loss range factor in the regret bounds as well. There has been much work on creating efficient algorithms that implicitly encode the weights over the set of combinatorial objects using concise representations. For example, many distributions over the 2^n subsets of n elements can be encoded by the probability of including each of the n elements. In addition to subsets, such work includes permutations (Helmbold and Warmuth, 2009; Yasutake et al., 2011; Ailon, 2014), paths (Takimoto and Warmuth, 2003; Kuzmin and Warmuth, 2005), and k-sets (Warmuth and Kuzmin, 2008).

There are also some general tools for learning combinatorial concepts. The Component Hedge algorithm of Koolen et al. (2010) is a powerful generic technique when the implicit encodings are suitably simple. The Component Hedge algorithm 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. A limitation of Component Hedge is its projection step which is generally only computationally efficient when there are a small (polynomial) number of constraints on the implicit representations.

1. Huffman trees (Cormen et al., 2001) are binary trees which construct prefix codes (called Huffman codes) for data compression. The plaintext symbols are located at the leaves of the tree and the path from the root to each leaf defines the prefix code for the associated symbol.

Suehiro et al. (2012) introduced another projection-based algorithm which specializes the Component Hedge algorithm for structures that can be formulated by submodular functions[2]. There are also projection-free algorithms for online learning. *Follow the Perturbed Leader (FPL)* (Kalai and Vempala, 2005) and its generalization (Dudík et al., 2017) are based on adding random perturbations to the cumulative loss of each component, and then predicting with the combinatorial object with minimum perturbed loss. Hazan and Kale (2012) introduced a projection-free online algorithm using the Frank-Wolfe technique.

The problem of concisely specifying the convex hulls of complicated combinatorial structures (e.g. permutations and Huffman trees) using few constraints has been well studied in the combinatorial optimization literature. A powerful technique – namely *extended formulations* – has been developed to represent these polytopes as a linear projection of a higher-dimensional polyhedron so that the polytope description has far fewer (polynomial instead of exponential) constraints (Kaibel and Pashkovich, 2013; Kaibel, 2011; Conforti et al., 2010).

Contributions: The main contributions of this paper are:

1. The introduction of extended formulation techniques to the machine learning community. In particular, the fusion of Component Hedge with extended formulations results in a new methodology for designing efficient online algorithms for complex classes of combinatorial objects. Our methodology uses a redundant representation 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. We are unaware of previous online learning work exploiting this kind of redundancy. 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. Component Hedge applications usually predict by first re-expressing the algorithm's weight or usage vector as a small convex combination of combinatorial objects, and then randomly sample from the convex combination. The redundant representation often allows for a more direct and efficient way to generate the algorithm's random prediction, bypassing the need to create convex combinations. This is always the case for extended formulations based on "reflection relations" (as in permutations and Huffman Trees).

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. The initialization of the redundant representation is more delicate. 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. This style of implicit initialization improves bounds in some existing work (e.g. saving a $\log n$ factor in Yasutake et al. (2011)) and has been used to good effect in another domain (Rahmanian and Warmuth, 2017).

Paper Outline: Section 2 contains an overview of the Component Hedge algorithm and extended formulations. Section 3 explains our methodology. We then explore the concrete application of our method on Huffman trees and permutation using extended formulations constructed by reflection relations in Section 4. Section 5 describes our fast prediction technique in the case of using reflection

2. For instance, permutations belong to such classes of structures (see Suehiro et al. (2012)); but Huffman trees do not as the sum of the code lengths of the symbols is not fixed.

relations. Finally, Section 6 concludes with contrasting our bounds with those of FPL (Kalai and Vempala, 2005), Hedge (Freund and Schapire, 1997) and OnlineRank (Ailon, 2014) and describing directions for future work. The Appendix A contains a summary of our notations.

2. Background

Online learning is a rich and vibrant area, see Cesa-Bianchi and Lugosi (2006) for a textbook treatment. The implicit representations for structured concepts (sometimes called 'indirect representations') have been used for a variety of problems (Helmbold et al., 2002; Helmbold and Schapire, 1997; Maass and Warmuth, 1998; Takimoto and Warmuth, 2002, 2003; Yasutake et al., 2011; Koolen et al., 2010). Recall from the Prediction Game 1, that $t \in \{1..T\}$ is the trial index, $\mathcal{H}$ is the class of combinatorial objects, $\widehat{\boldsymbol{h}}^{t-1} \in \mathcal{H}$ is the algorithm's selected object at time t, and $\boldsymbol{\ell}^t$ is the loss vector revealed by the adversary.

Component Hedge: Koolen et al. (2010) developed a generic framework called *Component Hedge* which results in efficient and effective online algorithms over combinatorial objects in $\mathbb{R}^n_+$ with linear loss. Component Hedge maintains a "usage" vector $\boldsymbol{v}$ in the polytope $\mathcal{V}$ which is the convex hull of all objects in the combinatorial class $\mathcal{H}$. In each trial, the weight of each component (i.e. coordinate) v_i of $\boldsymbol{v}$ is updated multiplicatively by its associated exponentiated loss: $v_i \leftarrow v_i \, e^{-\eta \, \ell_i}$.

Then the weight vector $\boldsymbol{v}$ is projected back to the polytope $\mathcal{V}$ via relative entropy projection. $\mathcal{V}$ is often characterized with a set of equality constraints (i.e. intersection of affine subspaces). Iterative Bregman projection (Bregman, 1967) is often used; it enforces each constraint in turn. Although this can violate previously satisfied constraints, repeatedly cycling through them is guaranteed to converge to the proper projection if all the facets of the polytope are equality constraints.

Finally, to sample with the same expectation as the usage vector, the usage vector is decomposed into corners of the polytope $\mathcal{V}$. Concretely, $\boldsymbol{v}$ is written as a convex combination of some objects in $\mathcal{H}$ using a greedy approach which zeros out at least one component in each iteration.

Component Hedge relies heavily on an efficient characterization of the polytope $\mathcal{V}$ both for projection and decomposition. If directly characterizing the polytope $\mathcal{V}$ is either difficult or requires exponentially many facets, Component Hedge cannot be directly applied. In those cases, we show how extended formulations can help with efficiently describing the polytope $\mathcal{V}$.

Extended Formulations: Many classes of combinatorial objects have polytopes whose discription requires exponentially many facets in the original space (e.g. see Maurras et al. (2010)). This has triggered the search for more concise descriptions in alternative spaces. In recent years, the combinatorial optimization community has given significant attention to the technique of *extended formulation* where difficult polytopes are represented as a linear projection of a higher-dimensional polyhedron (Magnanti and Wolsey, 1995; Conforti et al., 2010; Kaibel, 2011; Kaibel and Pashkovich, 2013). There are many complex combinatorial objects whose associated polyhedra can be described as the linear projection of a much simpler, but higher dimensional, polyhedra (see Figure 1).

Concretely, assume a polytope $\mathcal{V} \subset \mathbb{R}^n_+$ is given and described with exponentially many constraints in matrix-vector multiplication form as $\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}^n_+ \mid M_1 \boldsymbol{v} \leq \boldsymbol{d}\}$ in the original space $\mathbb{R}^n_+$. We assume that using some additional variables $\boldsymbol{x} \in \mathbb{R}^m_+$, $\mathcal{V}$ can be written efficiently as

$$\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}^n_+ \mid \exists \boldsymbol{x} \in \mathbb{R}^m_+ : M_2 \boldsymbol{v} + M_3 \boldsymbol{x} \leq \boldsymbol{f}\}^3 \tag{1}$$

3. Note that for each $\boldsymbol{v} \in \mathcal{V}$ there exists a $\boldsymbol{x} \in \mathcal{X}$, but it is not necessarily unique.

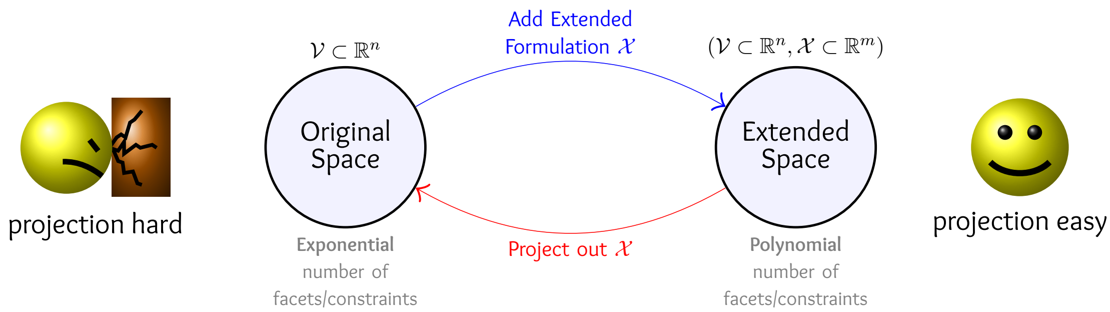


Figure 1: Extended formulation.

with $r = \text{poly}(n)$ constraints. Vector $\boldsymbol{x} \in \mathbb{R}^m_+$ is an e**x**tended formulation[4] belonging to the set

$$\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^m_+ \mid \exists \boldsymbol{v} \in \mathbb{R}^n_+ : M_2\boldsymbol{v} + M_3\boldsymbol{x} \leq \boldsymbol{f}\} \tag{2}$$

Extended formulations incur the cost of additional variables for the benefit of a simpler (although, higher dimensional) polytope.

3. The Method

Here we describe our general methodology for using extended formulations to develop new learning algorithms. Consider a class $\mathcal{H}$ of combinatorial objects and its convex hull $\mathcal{V}$. We assume there is no efficient description of $\mathcal{V}$ in $\mathbb{R}^n_+$, but it can be efficiently characterized via an extended formulation $\boldsymbol{x} \in \mathcal{X}$ as in Equations (1) and (2).

As described in Section 2, in order to apply Component Hedge (especially the projection), we need to have equality constraints instead of inequality ones. Thus, we introduce slack variables $\boldsymbol{s} \in \mathbb{R}^r_+$, where r is the number of constraints. Equation (1) now becomes

$$\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}^n_+ \mid \exists \boldsymbol{x} \in \mathbb{R}^m_+, \boldsymbol{s} \in \mathbb{R}^r_+ : M_2\boldsymbol{v} + M_3\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{f}\}$$

Now, in order to keep track of a usage vector $\boldsymbol{v} \in \mathcal{V}$, we use the following novel representation:

$$\mathcal{W} = \{\underbrace{(\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{s})}_{\boldsymbol{w}} \in \mathbb{R}^{n+m+r}_+ \mid M_2\boldsymbol{v} + M_3\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{f}\}$$

where $\mathcal{W}$ is characterized by r affine constraints. We refer to $\mathcal{W}$ as *the augmented formulation*. Observe that, despite potential redundancy in representation, all three constituents are useful in this new encoding: $\boldsymbol{v}$ is needed to encode the right loss, $\boldsymbol{x}$ is used for efficient description of the polytope, and $\boldsymbol{s}$ is incorporated to have equality constraints.

3.1 XF-Hedge Algorithm

Having developed the well-equipped space $\mathcal{W}$, Component Hedge can now be applied. Since $\boldsymbol{v}$ is the only constituent over which the loss vector $\boldsymbol{\ell}^t$ is defined, we work with $\boldsymbol{L}^t = (\boldsymbol{\ell}^t, \boldsymbol{0}, \boldsymbol{0}) \in [0,1]^{n+m+r}$ in the augmented formulation space $\mathcal{W}$. We introduce a new type of Hedge algorithm combined with e_x_tended _f_ormulation – *XF-Hedge* (See Algorithm 2). Similar to Component Hedge, XF-Hedge consists of three main steps: *Prediction*, *Update*, and *Projection*.

4. Throughout this paper, w.l.o.g., we assume $\boldsymbol{x}$ is in positive quadrant of $\mathbb{R}^n$, since an arbitrary point in $\mathbb{R}^n$ can be written as $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$ where $\boldsymbol{x}^+, \boldsymbol{x}^- \in \mathbb{R}^n_+$.

Algorithm 2 XF-Hedge

```
w^0 = (v^0, x^0, s^0) ∈ W – a proper prior distribution discussed in 3.2
For t = 1, ..., T
    Set ĥ^{t-1} ← Prediction(w^{t-1}) where ĥ^{t-1} ∈ H is a random object s.t. E[ĥ^{t-1}] = v^{t-1}
    Incur a loss ĥ^{t-1} · ℓ^t
    Update:
        Set ṽ_i^{t-1} ← v_i^{t-1} e^{-η ℓ_i^t} for all i ∈ [n]
    Set w^t ← Projection(ṽ^{t-1}, x^{t-1}, s^{t-1}) [= w̃^{t-1}] where w^t = arg min_{w∈W} Δ(w||w̃^{t-1})
```

Prediction: Randomly select an object $\widehat{\boldsymbol{h}}^{t-1}$ from the combinatorial class $\mathcal{H}$ in such a way that $\mathbb{E}\left[\widehat{\boldsymbol{h}}^{t-1}\right] = \boldsymbol{v}^{t-1}$. The details of this step depend on the combinatorial class $\mathcal{H}$ and the extended formulation used for $\mathcal{W}$. In Component Hedge and similar algorithms (Helmbold and Warmuth, 2009; Koolen et al., 2010; Yasutake et al., 2011; Warmuth and Kuzmin, 2008), this step is usually done by decomposing[5] $\boldsymbol{v}^{t-1}$ into a convex combination of objects in $\mathcal{H}$. In Section 5, we present a faster $\mathcal{O}(m+n)$ prediction method for combinatorial classes $\mathcal{H}$ whose extended formulation is constructed by reflection relations.

Update: Having defined $\boldsymbol{L}^t = (\boldsymbol{\ell}^t, \mathbf{0}, \mathbf{0})$, the updated $\widetilde{\boldsymbol{w}}^{t-1}$ is obtained using a trade-off between the linear loss and the unnormalized relative entropy (Koolen et al., 2010):

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

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

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

Thus this step takes $\mathcal{O}(n)$ time.

Projection: We use an unnormalized relative entropy Bregman projection to project $\widetilde{\boldsymbol{w}}^{t-1}$ back into $\mathcal{W}$ obtaining the new $\boldsymbol{w}^t$ for the next trial.

$$\boldsymbol{w}^t = \underset{\boldsymbol{w}\in\mathcal{W}}{\arg\min}\ \Delta(\boldsymbol{w}||\widetilde{\boldsymbol{w}}^{t-1}) \tag{3}$$

Let $\Psi_0, \ldots, \Psi_{r-1}$ be the r hyperplanes where the r constraints of $M_2\boldsymbol{v} + M_3\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{f}$ are satisfied, i.e. $\boldsymbol{w} \in \Psi_k$ if and only if $\boldsymbol{w}$ satisfies the kth constraint. Then $\mathcal{W}$ is the intersection of the Ψ_k's. Since the non-negativity constraints are already enforced by the definition of $\Delta(\cdot||\cdot)$, it is possible to solve (3) using iterative Bregman projections[6] (Bregman, 1967). Starting from $\boldsymbol{p}_0 = \widetilde{\boldsymbol{w}}^{t-1}$, we iteratively compute:

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

5. Note that according to Caratheodory's theorem, such decomposition exists in $\mathcal{W}$ using at most $n+m+r+1$ objects (i.e. corners of the polytope $\mathcal{W}$).
6. In Helmbold and Warmuth (2009) Sinkhorn balancing is used for projection which is also a special case of iterative Bregman projection.

repeatedly cycling through the constraints. It is known that $\boldsymbol{p}_k$ converges in Euclidean norm to the unique solution of (3) (Bregman, 1967; Bauschke and Borwein, 1997).

The projection step dominates the running time of the algorithm[7]. Projecting onto any of the r hyperplanes reduces to finding the sole non-negative (real) zero of a univariate polynomial of degree at most n (see Appendix F) so Newton's method takes $\mathcal{O}(n \log\log(1/\epsilon_1))$ time to get an ϵ_1-close solution. With r constraints ($r \in \mathcal{O}(n \log n)$ for our applications), each cycle through the constraints takes $\mathcal{O}(r\, n \log\log(1/\epsilon))$ time. Letting C_ϵ be the number of cycles to have an ϵ-accurate projection, the whole projection step takes $\mathcal{O}(C_\epsilon\, rn \log\log(1/\epsilon_1))$ time. These kinds of cyclic Bregman projections are believed to have fast linear convergence (Dhillon and Tropp, 2007), and empirically are very efficient (Koolen et al., 2010). Note that exact convergence is not essential. For example, if the projection step estimates each $\boldsymbol{w}^t$ within $\epsilon = 1/6n^2\sqrt{mT}$ then the additional loss over the entire sequence of T trials is less than 1 unit (see Appendix H). Therefore, with the linear convergence assumption, the projection step takes $\mathcal{O}(r \log(nmT) n \log\log(1/\epsilon_1))$ time.

3.2 Regret Bounds

Similar to Component Hedge, the general regret bound depends on the initial weight vector $\boldsymbol{w}^0 \in \mathcal{W}$ via $\Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0)$ where $\boldsymbol{w}(\boldsymbol{h}) \in \mathcal{W}$ is the augmented formulation of the object $\boldsymbol{h} \in \mathcal{H}$ against which the algorithm is compared (the best $\boldsymbol{h}$ for the adversarially chosen sequence of losses).

The following Lemma is somewhat optimistic that optimal tuning of η requires knowledge of the loss of the best $\boldsymbol{h}$ as well as its distance from the initial $\boldsymbol{w}^0$. However only slightly worse bounds can be achieved with doubling tricks to handle the unknown loss (see, e.g. Cesa-Bianchi and Lugosi (2006)) and a smart initialization of $\boldsymbol{w}_0$ provides an upper bound on $\Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0)$ for any object $\boldsymbol{h} \in \mathcal{H}$.

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

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

The proof uses standard techniques from the online learning literature (see, e.g. , Koolen et al. (2010)) and is given in Appendix B. In order to get good bounds, the initial weight $\boldsymbol{w}^0$ must be "close" to all corners $\boldsymbol{h}$ of the polytope, and thus in the"middle" of $\mathcal{W}$. In previous works (Koolen et al., 2010; Yasutake et al., 2011; Helmbold and Warmuth, 2009), the initial weight is explicitly chosen and it is often set to be the uniform usage of the objects. This explicit initialization approach may be difficult to perform when the polytope has a complex structure.

Here, instead of explicitly selecting $\boldsymbol{w}^0 \in \mathcal{W}$, we implicitly design the initial point. First, we find an intermediate "middle" point $\widetilde{\boldsymbol{w}} \in \mathbb{R}^{n+m+r}$ with good distance properties, and then project $\widetilde{\boldsymbol{w}}$ into $\mathcal{W}$ to obtain the initial $\boldsymbol{w}^0$ for the first trial.

A good choice for $\widetilde{\boldsymbol{w}}$ is $U\,\mathbf{1}$ where $\mathbf{1} \in \mathbb{R}^{n+m+r}$ is the vector of all ones, and $U \in \mathbb{R}_+$ is an upper-bound on the infinity norms of the corners of polytope $\mathcal{W}$. This leads to the nice bound $\Delta(\boldsymbol{w}(\boldsymbol{h})||\widetilde{\boldsymbol{w}}) \leq (n + m + r)U$ for all objects $\boldsymbol{h} \in \mathcal{H}$. The Generalized Pythagorean Theorem (Herbster and Warmuth, 2001) ensures that the same bound holds for $\boldsymbol{w}^0$ (see Appendix C for the details).

7. If the model is being trained on data where predictions are not required, then the expensive projection step can be deferred until the predictions are needed.

Lemma 2 *Assume that there exists $U \in \mathbb{R}_+$ such that $\|\boldsymbol{w}(\boldsymbol{h})\|_\infty \leq U$ for all $\boldsymbol{h} \in \mathcal{H}$. Then the initialization method finds a $\boldsymbol{w}^0 \in \mathcal{W}$ such that for all $\boldsymbol{h} \in \mathcal{H}$, $\Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0) \leq (n+m+r)\,U$.*

Combining Lemmas 1, and 2 gives the following guarantee.

Theorem 3 *If each $\boldsymbol{\ell}^t \in [0,1]^n$ and $\|\boldsymbol{w}(\boldsymbol{h})\|_\infty \leq U$ for all $\boldsymbol{h} \in \mathcal{H}$, then XF-hedge's regret is:*

$$\mathbb{E}\left[\sum_{t=1}^{T} \widehat{\boldsymbol{h}}^{t-1} \cdot \boldsymbol{\ell}^t\right] - \min_{\boldsymbol{h}\in\mathcal{H}} \sum_{t=1}^{T} \boldsymbol{h} \cdot \boldsymbol{\ell}^t \leq \sqrt{2L^*\,(n+m+r)\,U} + (n+m+r)\,U$$

4. XF-Hedge Examples Using Reflection Relations

One technique for constructing extended formulations is called *reflection relations* (Kaibel and Pashkovich, 2013), and this technique can be used to efficiently describe the polytopes of permutations and Huffman trees. Here we describe how reflection relations can be used with the XF-Hedge framework to create concrete learning algorithms for permutations and Huffman trees.

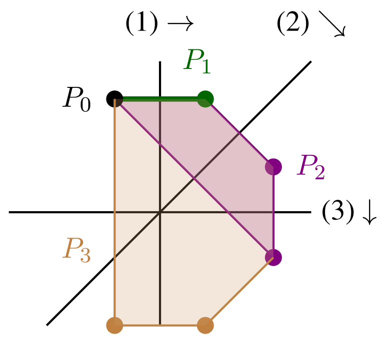
As in Yasutake et al. (2011) and Ailon (2014), we consider losses that are linear in the first order representation of the objects (Diaconis, 1988). For permutations of n items, the first order representation is vectors $\boldsymbol{v} \in \mathbb{R}^n$ where each of the elements of $\{1, 2, \ldots, n\}$ appears exactly once[8] and for Huffman trees on n symbols, the first order representation is vectors $\boldsymbol{v} \in \mathbb{R}^n$ where each v_i is an integer indicating the depth of the leaf corresponding to symbol i in the coding tree. At each trial the loss is $\boldsymbol{v} \cdot \boldsymbol{\ell}$ where the adversary's $\boldsymbol{\ell}$ is a loss vector in the unit cube $[0,1]^n$. This type of loss is sufficiently rich to capture well-known natural losses like *average code length* for Huffman trees (when ℓ is the symbol frequencies) and *sum of completion times* for permutations[9] (when ℓ is the task completion times).

Constructing Extended Formulations from Reflection Relations Kaibel and Pashkovich (2013) show how to construct polynomial size extended formulations using a canonical corner of the polytope and a fixed sequence of hyperplanes. These have the property that any corner of the desired polytope can be generated by reflecting the canonical corner through a subsequence of the hyperplanes. These reflections are *one-sided* in the sense that they map the half-space containing the canonical corner to the other half-space. For example, the corners of Figure 2 (Left) can be generated in this way. Of course the hard part is to find a good sequence of hyperplanes with this property.

A key idea for generating the entire polytope is to allow "partial reflections" where the point to be reflected can not just be kept (skipping the reflection) or replaced by its reflected image, but mapped to any point on the line segment joining the point and its reflected image as illustrated in Figure 2 (Right). Since any point in the convex hull of the polytope can be constructed by at least one sequence of partial one-sided reflections, every point in the polytope has an alternative representation in terms of how much each reflection was used to generate it from the canonical corner (see Figure 2). Each of these parameterized partial reflections is a *reflection relation*,

For each reflection relation, there will be one additional variable indicating the extent to which the reflection occurs, and two additional inequalities for the extreme cases of complete reflection

8. In contrast, Helmbold and Warmuth (2009) work with the second order representation (i.e. Birkhoff polytope), and consequently losses, which is a more general loss family (see Yasutake et al. (2011) for comparison).
9. To easily encode the sum of completion times, the predicted permutation represents the *reverse* order in which the tasks are to be executed.



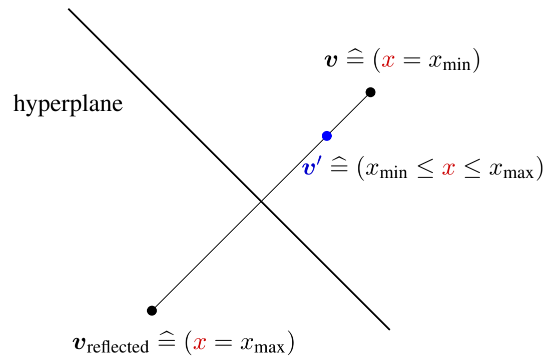


Figure 2: (Left) The 6 corners of the polytope are generated by subsequences of one-sided reflections through lines (1), (2), and (3), starting from the canonical point P_0. Using partial reflections, we can generate the entire polytope. (Right) A partial reflection of $\boldsymbol{v}$ to $\boldsymbol{v}'$ corresponds to ($\widehat{=}$) a variable x indicating how far $\boldsymbol{v}'$ moves towards $\boldsymbol{v}$'s image $\boldsymbol{v}_{\text{reflected}}$.

and no reflection. Therefore, if the polytope can be expressed with polynomially many reflection relations, then it has an extended formulation of polynomial size with polynomially many constraints. Appendix D provides more details about the type of results shown by Kaibel and Pashkovich (2013).

Extended Formulations for Objects Closed under Re-Ordering Assume we want to construct an extended formulation for a class of combinatorial objects which is closed under any re-ordering (both Huffman trees and permutations both have this property). Then reflection relations corresponding to swapping pairs of elements are useful. Swapping elements i and j can be implemented with a hyperplane going through the origin and having normal vector $\boldsymbol{e}_i - \boldsymbol{e}_j$ (here $\boldsymbol{e}_i$ is the ith unit vector). The identity permutation is the natural canonical corner, so the one-sided reflections are only used for $\boldsymbol{v}$ where $v_i \le v_j$.

Implementing the reflection relation for the i, j swap uses an additional variable along with two additional inequalities. Concretely, assume $\boldsymbol{v} \in \mathbb{R}^n$ is going into this reflection relation and $\boldsymbol{v}' \in \mathbb{R}^n$ is the output, so $\boldsymbol{v}'$ is in the convex combination of $\boldsymbol{v}$ and its reflection. It is natural to encode this as $\boldsymbol{v}' = \gamma \boldsymbol{v} + (1-\gamma)\boldsymbol{v}_{\text{reflected}}$. However, we found it more convenient to parameterize $\boldsymbol{v}'$ by its absolute distance x from $\boldsymbol{v}$, rather than the relative distance $\gamma \in [0,1]$. Using this parameterization, we have $\boldsymbol{v}' = \boldsymbol{v} + x\,(\boldsymbol{e}_i - \boldsymbol{e}_j)$ constrained by $(\boldsymbol{e}_i - \boldsymbol{e}_j)\cdot \boldsymbol{v} \le (\boldsymbol{e}_i - \boldsymbol{e}_j)\cdot \boldsymbol{v}' \le -(\boldsymbol{e}_i - \boldsymbol{e}_j)\cdot \boldsymbol{v}$. Therefore the possible relationships between between $\boldsymbol{v}'$ and $\boldsymbol{v}$ can be encoded with the additional variable x and the following constraints:[10]

$$\boldsymbol{v}' = \boldsymbol{m}\,x + \boldsymbol{v} \quad \text{where } \boldsymbol{m} = \boldsymbol{e}_i - \boldsymbol{e}_j, \qquad 0 \le x \le v_j - v_i. \tag{4}$$

Notice that x indicates the amount of change in the ith and jth elements which can go from zero (remaining unchanged) to the maximum swap capacity $v_j - v_i$.

Suppose the desired polytope is described using m reflection relations and with canonical point $\boldsymbol{c}$. Then starting from $\boldsymbol{c}$ and successively applying the equation in (4), we obtain the connection between the extended formulation space $\mathcal{X}$ and original space $\mathcal{V}$:

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

10. In general $\boldsymbol{v}$ (and thus v_j and v_i) may be functions of the variables for previous reflection relations.

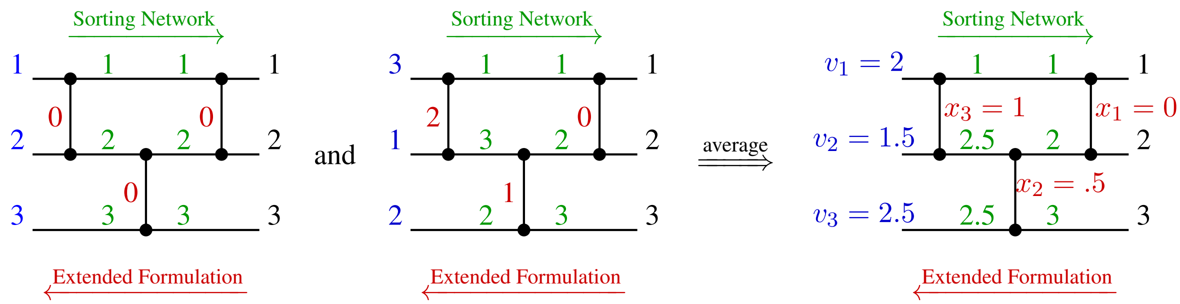


Figure 3: An extended formulation for permutation on $n = 3$ items. The canonical permutation is $[1, 2, 3]$. Elements of $\boldsymbol{v}$ are in blue, $\boldsymbol{x}$ in red, and the intermediate values are in green.

Kaibel and Pashkovich (2013) showed that the m reflection relations corresponding to the m comparators in an arbitrary n-input sorting network[11] generates the permutation polytope (see Figure 3). A similar extended formulation for Huffman trees can be built using an arbitrary sorting network along with $O(n \log n)$ additional comparators and simple linear maps (which do not require extra variables) and the canonical corner $\boldsymbol{c} = [1, 2, \ldots, n-2, n-1, n-1]^T$ (see Section 2.24 in Pashkovich (2012) for more details). Note that the reflection relations are applied in reverse order than their use in the sorting network (see Figure 3).

Learning Permutations and Huffman Trees As described in the previous subsections, the polytope $\mathcal{V}$ of both permutations and Huffman trees can be efficiently described using m inequality and n equality constraints[12]:

$$\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}_+^n | \exists \boldsymbol{x} \in \mathbb{R}_+^m : A\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{v} = M\boldsymbol{x} + \boldsymbol{c}\}$$

Adding the slack variables $\boldsymbol{s} \in \mathbb{R}_+^m$, we obtain the augmented formulation $\mathcal{W}$:

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

Note that all the wire values (i.e. v_i's), as well as x_i's and s_i's are upperbounded by $U = n$. Using the AKS sorting networks with $m = \mathcal{O}(n \log n)$ comparators (Ajtai et al., 1983), we can obtain the regret bounds below from Theorem 3:

Corollary 4 *XF-Hedge has the following regret bound when learning either permutations or Huffman trees:*

$$\mathbb{E}\left[\sum_{t=1}^{T} \widehat{\boldsymbol{h}}^{t-1} \cdot \boldsymbol{\ell}^t\right] - \min_{\boldsymbol{h} \in \mathcal{H}} \sum_{t=1}^{T} \boldsymbol{h} \cdot \boldsymbol{\ell}^t = \mathcal{O}\left(n\,(\log n)^{\frac{1}{2}} \sqrt{L^*} + n^2 \log n\right)$$

5. Fast Prediction with Reflection Relations

From its current weight vector $\boldsymbol{w} = (\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{s}) \in \mathcal{W}$, XF-Hedge randomly selects an object $\widehat{\boldsymbol{h}}$ from the combinatorial class $\mathcal{H}$ in such a way that $\mathbb{E}\left[\widehat{\boldsymbol{h}}\right] = \boldsymbol{v}$. In Component Hedge and similar

11. A *sorting network* is a sorting algorithm where the comparisons are fixed in advance. See e.g. (Cormen et al., 2001).
12. Positivity constraints are excluded as they are already enforced due to definition of $\Delta(\cdot||\cdot)$, and Huffman trees require additional $\mathcal{O}(n \log n)$ inequality constraints beyond those corresponding to the sorting network.

algorithms (Helmbold and Warmuth, 2009; Koolen et al., 2010; Yasutake et al., 2011; Warmuth and Kuzmin, 2008), this is done by decomposing $\boldsymbol{v}$ into a convex combination of objects in $\mathcal{H}$ followed by sampling. In this section, we give a new more direct prediction method for combinatorial classes $\mathcal{H}$ whose extended formulation is constructed by reflection relations. Our method is faster due to avoiding the decomposition.

The values $\boldsymbol{x}$ and $\boldsymbol{x}+\boldsymbol{s}$ can be interpreted as amount swapped and the maximum swap allowed for the comparators in the sorting networks, respectively. Therefore, it is natural to define $x_i/(x_i+s_i)$ as *swap probability* associated with the ith comparator for $i \in \{1..m\}$. Algorithm 3 incorporates the notion of swap probabilities to construct an efficient sampling procedure from a distribution $\mathcal{D}$ which has the right expectation. It starts with the canonical object (e.g. identity permutation) and feeds it through the reflection relations. Each reflection i is taken with probability $x_i/(x_i+s_i)$. The theorem below (proved in the Appendix G) guarantees the correctness and efficiency of this algorithm.

Theorem 5 *(i) Given $(\boldsymbol{v},\boldsymbol{x},\boldsymbol{s}) \in \mathcal{W} \subseteq \mathbb{R}^{n+2m}$, Algorithm 3 samples $\boldsymbol{h}$ from a distribution $\mathcal{D}$ such that $\mathbb{E}_{\mathcal{D}}[\boldsymbol{h}] = \boldsymbol{v}$.*
(ii) The time complexity of Algorithm 3 is $\mathcal{O}(n+m)$.

Using the AKS sorting networks (Ajtai et al., 1983), $m \in \mathcal{O}(n \log n)$, so Algorithm 3 predicts in $\mathcal{O}(n \log n)$ time. This improves the previously known $\mathcal{O}(n^2)$ prediction procedure for mean-based algorithms[13] for permutations (Yasutake et al., 2011; Suehiro et al., 2012).

Algorithm 3 Fast-Prediction

Input: $(\boldsymbol{x},\boldsymbol{s}) \in \mathbb{R}_+^{2m}$
Output: A prediction $\widehat{\boldsymbol{h}} \in \mathcal{H} \subseteq \mathbb{R}^n$
$\widehat{\boldsymbol{h}} \leftarrow \boldsymbol{c}$ *– the canonical corner in* $\mathbb{R}^n$
for $k = 1$ to m **do**
  $(i_k, j_k) \leftarrow$ wire indices associated with the k-th comparator
  if $x_i = 0$ **then**
    continue
  else
    Switch the i_kth and j_kth components of $\widehat{\boldsymbol{h}}$ w.p. $x_k/(x_k+s_k)$.
return $\widehat{\boldsymbol{h}}$

6. Conclusion and Future Work

Table 1 contains a comparison of the regret bounds for the new XF-Hedge algorithm, OnlineRank (Ailon, 2014), Follow the Perturbed Leader (FPL) (Kalai and Vempala, 2005), and the Hedge algorithm (Freund and Schapire, 1997) which inefficiently maintains an explicit weight for each of the exponentially many permutations or Huffman trees. For permutations, the regret bound of general XF-Hedge methodology is within a factor $\sqrt{\log n}$ of the state-of-the-art algorithm OnlineRank

13. It also matches the time complexity of prediction step for non-mean-based permutation-specialized OnlineRank (Ailon, 2014) and also the general FPL (Kalai and Vempala, 2005) algorithm both for permutations and Huffman Trees.

Algorithm	Permutation	Huffman Tree	
		$\boldsymbol{\ell}^t \in$ Unit Cube	$\boldsymbol{\ell}^t \in$ Unit Simplex
XF-Hedge	$\mathcal{O}\begin{pmatrix} n(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^2 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^2 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^2 \log n \end{pmatrix}$
OnlineRank	$\mathcal{O}(n\sqrt{L^*} + n^2)$	–	–
FPL	$\mathcal{O}\begin{pmatrix} n^{\frac{3}{2}}(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^3 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n^{\frac{3}{2}}(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^3 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^2 \log n \end{pmatrix}$
Hedge Algorithm	$\mathcal{O}\begin{pmatrix} n^{\frac{3}{2}}(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^3 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n^{\frac{3}{2}}(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^3 \log n \end{pmatrix}$	$\mathcal{O}\begin{pmatrix} n(\log n)^{\frac{1}{2}}\sqrt{L^*} \\ +n^2 \log n \end{pmatrix}$

Table 1: Comparing the regret bounds of XF-Hedge with other existing algorithms in different problems and different loss regimes.

(Ailon, 2014). When compared with the generic explicit Hedge algorithm (which is not computationally efficient) and FPL, XF-Hedge has a better loss bound by a factor of $\sqrt{n}$.

When comparing on Huffman trees, we consider two loss regimes: one where the loss vectors are from the general unit cube, and consequently, the per-trial losses are in $\mathcal{O}(n^2)$ (like permutations), and another where the loss vectors represent frequencies and lie on the unit simplex so the per-trial losses are in $\mathcal{O}(n)$. In the first case, as with permutations, XF-Hedge has the best asymptotic bounds. In the second case, the lower loss range benefits Hedge and FPL, and the regret bounds of all three algorithms match.

In conclusion, we have presented a general methodology for creating online learning algorithms from extended formulations. Our main contribution is the XF-Hedge algorithm that enables the efficient use of Component Hedge techniques on complex classes of combinatorial objects. Because XF-Hedge is in the Bregman projection family of algorithms, many of the tools from the expert setting are likely to carry over. This includes lower bounding weights for shifting comparators (Herbster and Warmuth, 1998), long-term memory (Bousquet and Warmuth, 2002), and adapting the updates to the bandit setting (Audibert et al., 2011). Several important areas remain for potentially fruitful future work:

More Applications There is a rich literature on extended formulation for different combinatorial objects (Conforti et al., 2010; Kaibel, 2011; Pashkovich, 2012; Afshari Rad and Kakhki, 2017; Fiorini et al., 2013). Which combinatorial classes have both natural online losses and suitable extended formulations so XF-Hedge is appropriate? For instance, building on the underlying ideas of XF-Hedge, Rahmanian and Warmuth (2017) developed a family of learning algorithms focusing on extended formulations constructed by dynamic programming.

More Complex Losses The redundant representation we introduced can be used to express different losses. Although our current applications do not assign loss to the extended formulation variables ($\boldsymbol{x}$) and their associated slack variables ($\boldsymbol{s}$), these additional variables enable the expression of different kinds of losses. For what natural losses could be these additional variables useful?

Acknowledgments

We thank Manfred K. Warmuth for his helpful discussions. We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-1619271).

References

Maria Afshari Rad and Hossein Taghizadeh Kakhki. Two extended formulations for cardinality maximum flow network interdiction problem. *Networks*, 2017.

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

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

Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Minimax policies for combinatorial prediction games. In *COLT*, volume 19, pages 107–132, 2011.

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.

Olivier Bousquet and Manfred K Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3(Nov):363–396, 2002.

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.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR: A Quarterly Journal of Operations Research*, 8(1):1–48, 2010.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition, 2001.

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.

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

Miroslav Dudík, Nika Haghtalab, Haipeng Luo, Robert E. Schapire, Vasilis Syrgkanis, and Jennifer Wortman Vaughan. Oracle-efficient learning and auction design. In *Proceedings of 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 2017.

Samuel Fiorini, Volker Kaibel, Kanstantsin Pashkovich, and Dirk Oliver Theis. Combinatorial bounds on nonnegative rank and extended formulations. *Discrete mathematics*, 313(1):67–83, 2013.

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.

Elad Hazan and Satyen Kale. Projection-free online learning. In *Proceedings of 29th International Conference on Machine Learning (ICML)*, 2012.

David P Helmbold and Robert E Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, 1997.

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

David P Helmbold, Sandra Panizza, and Manfred K Warmuth. Direct and indirect algorithms for on-line learning of disjunctions. *Theoretical Computer Science*, 284(1):109–142, 2002.

Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine Learning*, 32(2): 151–178, 1998.

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

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

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

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

Wouter M. Koolen, Manfred K. Warmuth, and Jyrki Kivinen. Hedging structured concepts. In *Proceedings of COLT*, pages 93–105, 2010.

Dima Kuzmin and Manfred K Warmuth. Optimum follow the leader algorithm. In *Learning Theory*, pages 684–686. Springer, 2005.

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

Wolfgang Maass and Manfred K Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

Thomas L Magnanti and Laurence A Wolsey. Optimal trees. *Handbooks in operations research and management science*, 7:503–615, 1995.

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.

Kanstantsin Pashkovich. *Extended formulations for combinatorial polytopes*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2012.

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

Daiki Suehiro, Kohei Hatano, Shuji Kijima, Eiji Takimoto, and Kiyohito Nagano. Online prediction under submodular constraints. In *International Conference on Algorithmic Learning Theory*, pages 260–274. Springer, 2012.

Eiji Takimoto and Manfred K Warmuth. Predicting nearly as well as the best pruning of a planar decision graph. *Theoretical Computer Science*, 288(2):217–235, 2002.

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

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.

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.

Appendix A. Table of Notations

Symbol	Description
n	The dimensionality of the combinatorial object
$\mathcal{H}$	The set of all objects in $\mathbb{R}^n_+$
$\boldsymbol{h}$	A particular object in $\mathcal{H}$
T	The number of trials
$\boldsymbol{\ell}$	The loss vector revealed by the adversary in $[0,1]^n$
$\mathcal{V}$	The convex hull of all objects in $\mathcal{H}$
$\boldsymbol{v}$	A point in $\mathcal{H}$
m	The dimensionality of the extended formulation
$\boldsymbol{x}$	A point in extended formulation
$\mathcal{X}$	The space of extended formulations $\boldsymbol{x}$
$\mathcal{W}$	The augmented formulation
$\boldsymbol{w}$	A point in the augmented formulation $\mathcal{W}$
$\boldsymbol{s}$	The slack vector in the augmented formulation
r	The dimensionality of the slack vector
$\Delta(\cdot\|\|\cdot)$	the unnormalized relative entropy i.e. $\Delta(\boldsymbol{w}_1\|\|\boldsymbol{w}_2) = \sum_i w_{1,i} \log \frac{w_{1,i}}{w_{2,i}} + w_{2,i} - w_{1,i}$
$\boldsymbol{w}(\boldsymbol{h})$	A point in $\mathcal{W}$ associated with the object $\boldsymbol{h}$
U	An upper-bound for $\\|\boldsymbol{w}(\boldsymbol{h})\\|_\infty$
L^*	The cumulative loss of the best object in hindsight i.e. $\min_{\boldsymbol{h}\in\mathcal{H}} \sum_{t=1}^T \boldsymbol{h}\cdot\boldsymbol{\ell}^t$
M	the $n \times m$ matrix representing the affine transformation corresponding to m reflection relations
$\boldsymbol{c}$	the canonical point in $\mathcal{H}$ e.g. $[1,2,\ldots,n]^T$ for permutations
A, $\boldsymbol{b}$	the $m \times m$ matrix of coefficients and m-dimensional vector of constant terms specifying $\mathcal{X}$ along $\boldsymbol{x} \geq \boldsymbol{0}$ i.e. $A\boldsymbol{x} \leq \boldsymbol{b}$

Table 2: Table of notations in the order of appearance in the paper.

Appendix B. Proof of Lemma 1

Proof Assuming $\boldsymbol{w} = (\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{s})$ and $\boldsymbol{L} = (\boldsymbol{\ell}, \boldsymbol{0}, \boldsymbol{0})$:

$$\begin{aligned}(1-e^{-\eta})\boldsymbol{v}^{t-1}\cdot\boldsymbol{\ell}^t = (1-e^{-\eta})\boldsymbol{w}^{t-1}\cdot\boldsymbol{L}^t &\leq \sum_i w_i^{t-1}(1-e^{-\eta L_i^t}) \\ &= \Delta(\boldsymbol{w}(\boldsymbol{h})\|\boldsymbol{w}^{t-1}) - \Delta(\boldsymbol{w}(\boldsymbol{h})\|\widetilde{\boldsymbol{w}}^{t-1}) + \eta\, \boldsymbol{w}(\boldsymbol{h})\cdot\boldsymbol{L}^t \\ &= \Delta(\boldsymbol{w}(\boldsymbol{h})\|\boldsymbol{w}^{t-1}) - \Delta(\boldsymbol{w}(\boldsymbol{h})\|\widetilde{\boldsymbol{w}}^{t-1}) + \eta\, \boldsymbol{h}\cdot\boldsymbol{\ell}^t \\ &\leq \Delta(\boldsymbol{w}(\boldsymbol{h})\|\boldsymbol{w}^{t-1}) - \Delta(\boldsymbol{w}(\boldsymbol{h})\|\boldsymbol{w}^{t}) + \eta\, \boldsymbol{h}\cdot\boldsymbol{\ell}^t\end{aligned}$$

The first inequality is obtained using $1-e^{-\eta x} \geq (1-e^{-\eta})x$ for $x \in [0,1]$ as done in Littlestone and Warmuth (1994). The second inequality is a result of the Generalized Pythagorean Theorem (Herbster and Warmuth, 2001), since $\boldsymbol{w}^t$ is a Bregman projection of $\widehat{\boldsymbol{w}}^{t-1}$ into the convex set $\mathcal{W}$

which contains $\boldsymbol{w}(\boldsymbol{h})$. By summing over $t = 1 \ldots T$ and using the non-negativity of divergences, we obtain:

$$
\begin{aligned}
(1 - e^{-\eta}) \sum_{t=1}^{T} \boldsymbol{v}^{t-1} \cdot \boldsymbol{\ell}^t &\le \Delta(\boldsymbol{w}(\boldsymbol{h}) || \boldsymbol{w}^0) - \Delta(\boldsymbol{w}(\boldsymbol{h}) || \boldsymbol{w}^T) + \eta \sum_{t=1}^{T} \boldsymbol{h} \cdot \boldsymbol{\ell}^t \\
\longrightarrow \mathbb{E}\left[\sum_{t=1}^{T} \boldsymbol{h}^{t-1} \cdot \boldsymbol{\ell}^t\right] &\le \frac{\eta \sum_{t=1}^{T} \boldsymbol{h} \cdot \boldsymbol{\ell}^t + \Delta(\boldsymbol{w}(\boldsymbol{h}) || \boldsymbol{w}^0)}{1 - e^{-\eta}}
\end{aligned}
$$

Let $L^* = \min_{\boldsymbol{h} \in \mathcal{H}} \sum_{t=1}^{T} \boldsymbol{h} \cdot \boldsymbol{\ell}^t$. We can set the learning rate $\eta = \sqrt{\frac{2\,\Delta(\boldsymbol{w}(\boldsymbol{h})||\boldsymbol{w}^0)}{L^*}}$ as instructed in Koolen et al. (2010) and obtain the following regret bound:

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

Appendix C. Proof of Lemma 2

Proof Let $\widetilde{\boldsymbol{w}} = U\,\mathbf{1}$ in which $\mathbf{1} \in \mathbb{R}^{m+n+r}$ is a vector with all ones in its components. Now let $\boldsymbol{w}^0$ be the Bregman projection of $\widetilde{\boldsymbol{w}}$ onto $\mathcal{W}$, that is:

$$
\boldsymbol{w}^0 = \underset{\boldsymbol{w} \in \mathcal{W}}{\arg\min} \Delta(\boldsymbol{w} || \widetilde{\boldsymbol{w}})
$$

Now for all $\boldsymbol{h} \in \mathcal{H}$, we have:

$$
\begin{aligned}
\Delta(\boldsymbol{w}(\boldsymbol{h}) || \boldsymbol{w}^0) &\le \Delta(\boldsymbol{w}(\boldsymbol{h}) || \widetilde{\boldsymbol{w}}) && \text{Pythagorean Theorem} \\
&= \sum_{i \in \{1..n+m+r\}} (\boldsymbol{w}(\boldsymbol{h}))_i \log \frac{(\boldsymbol{w}(\boldsymbol{h}))_i}{U} + U - (\boldsymbol{w}(\boldsymbol{h}))_i \\
&\le \sum_{i \in \{1..n+m+r\}} U && (\boldsymbol{w}(\boldsymbol{h}))_i \le U \\
&= (n + m + r)\,U && \blacksquare
\end{aligned}
$$

Appendix D. Construction of Extended Formulation Using Reflection Relations

Instead of starting with a single corner, one could also consider passing an entire polytope as an input through the sequence of (partial) reflections to generate a new polytope. Using this fact, Theorem 1 in Kaibel and Pashkovich (2013) provides an inductive construction of higher dimensional polytopes via sequences of reflection relations. Concretely, let P^n_{obj} be the polytope of a given combinatorial object of size n. The typical approach is to properly embed $P^n_{\text{obj}} \subset \mathbb{R}^n$ into $\widehat{P}^n_{\text{obj}} \subset \mathbb{R}^{n+1}$, and then feed it through an appropriate sequence of reflection relations as an input polytope in order to obtain an extended formulation for $P^{n+1}_{\text{obj}} \subset \mathbb{R}^{n+1}$. Theorem 1 in Kaibel and Pashkovich (2013) provides sufficient conditions for the correctness of this procedure. Again, 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 P^n_{obj} with polynomially many constraints. In this paper, however, we work with batch construction of the extended formulation as opposed to the inductive construction.

Appendix E. Facets Constructed by Reflection Relations

Lemma 6 *Let M be the matrix representing the affine transformation corresponding to m reflection relations and*

$$A = \mathit{Tri}(M^T M) + I, \quad \boldsymbol{b} = -M^T \boldsymbol{c}$$

in which $\mathit{Tri}(\cdot)$ is a function over square matrices which zeros out the upper triangular part of the input including the diagonal. Then the extended formulation space $\mathcal{X}$ is

$$A\boldsymbol{x} \leq \boldsymbol{b}, \qquad \boldsymbol{x} \geq \boldsymbol{0}$$

or equivalently with slack variables $\boldsymbol{s}$

$$A\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}, \qquad \boldsymbol{x}, \boldsymbol{s} \geq \boldsymbol{0}$$

Proof Let $\boldsymbol{v}^k$ be the vector in $\mathcal{V}$ after going through the kth reflection relation. Also denote the kth column of M by M_k. Observe that $\boldsymbol{v}^0 = \boldsymbol{c}$ and $\boldsymbol{v}^k = \boldsymbol{c} + \sum_{i=1}^{k} M_i x_i$. Let $M_k = \boldsymbol{e}_r - \boldsymbol{e}_s$. Then, using (4), the inequality associated with the kth row of $A\boldsymbol{x} \leq \boldsymbol{b}$ will be obtained as below:

$$x_k \leq v_s^{k-1} - v_r^{k-1} = -M_k^T \boldsymbol{v}^{k-1} = -M_k^T \left(\boldsymbol{c} + \sum_{i=1}^{k-1} M_i x_i \right)$$

$$\longrightarrow x_k + \sum_{i=1}^{k-1} M_k^T M_i x_i \leq -M_k^T \boldsymbol{c} = b_k$$

Thus:

$$\forall\, i, j \in [m] \quad A_{ij} = \begin{cases} M_i^T M_j & i > j \\ 1 & i = j \\ 0 & i < j \end{cases}, \qquad \forall\, k \in [m] \quad b_k = -M_k^T \boldsymbol{c}$$

which concludes the proof. ■

Appendix F. Projection onto Each Constraint

Each constraint of the polytope in the augmented formulation is of the form $\boldsymbol{a}^T \boldsymbol{w} = a_0$. Formally, the projection $\boldsymbol{w}^*$ of a give point $\boldsymbol{w}$ to this constraint is solution to the following:

$$\underset{\boldsymbol{a}^T \boldsymbol{w}^* = a_0}{\arg\min} \sum_i w_i^* \log\left(\frac{w_i^*}{w_i}\right) + w_i - w_i^*$$

Finding the solution to the projection above for general hyperplanes and Bregman divergence can be found in Section 3 of Dhillon and Tropp (2007). Nevertheless, for the sake of completeness, we also provide the solution for the particular case of Huffman trees and permutations described by $\mathcal{W}$ in Section 4 as well. Using the method of Lagrange multipliers, we have:

$$L(\boldsymbol{w}^*, \mu) = \sum_i w_i^* \log\left(\frac{w_i^*}{w_i}\right) + w_i - w_i^* - \mu \left(\sum_{j=1}^{2m+n} a_i\, w_i^* - a_0 \right)$$

$$\frac{\partial L}{\partial w_i^*} = \log\left(\frac{w_i^*}{w_i}\right) - \mu a_i = 0, \quad \forall i \in [n + 2m]$$

$$\frac{\partial L}{\partial \mu} = \sum_{j=1}^{2m+n} a_i\, w_i^* - a_0 = 0$$

Replacing $\rho = e^{-\mu}$, we have $w_i^* = w_i\, \rho^{a_i}$. By enforcing $\frac{\partial L}{\partial \mu} = 0$, one needs to find $\rho > 0$ such that:

$$\sum_{i=1}^{n+2m} a_i\, w_i\, \rho^{a_i} - a_0 = 0 \tag{5}$$

Observe that due to the structure of matrices M and A (see Lemma 6), $a_i \in \mathbb{Z}$ and $a_i \geq -1$ for all $i \in [n + 2m]$, and furthermore $a_0 \geq 0$. Multiplying by ρ, we can re-write equation (5) as the polynomial below:

$$f(\rho) = \phi_k \rho^k + \ldots + \phi_2 \rho^2 - \phi_1 \rho - \phi_0 = 0$$

in which all ϕ_i's are non-negative real numbers and $k \leq n$. Note that $f(0) < 0$ and $f(\rho) \to +\infty$ as $\rho \to +\infty$. Thus $f(\rho)$ has at least one positive root. However, it can not have more than one positive roots and we can prove it by contradiction. Assume that there exist $0 < r_1 < r_2$ such that $f(r_1) = f(r_2) = 0$. Since f is convex on positive real line, using Jensen's inequality, we can obtain the contradiction below:

$$0 = f(r_1) = f\left(\frac{r_2 - r_1}{r_2} \times 0 + \frac{r_1}{r_2} \times r_2\right) < \frac{r_2 - r_1}{r_2} f(0) + \frac{r_1}{r_2} f(r_2) = \frac{r_2 - r_1}{r_2} f(0) < 0$$

Therefore f has exactly one positive root which can be found by Newton's method starting from a sufficiently large initial point. Note that if the constraint belongs to $\boldsymbol{v} = M\boldsymbol{x} + \boldsymbol{c}$, then all the a_i's are in $\{-1, 0, 1\}$ and polynomial $f(\rho)$ will be quadratic, so there is a closed form for the positive root.

Appendix G. Proof of Theorem 5

Proof Let $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$. Using induction, we prove that by the end of the ith loop of Algorithm 3, the obtained distribution $\mathcal{D}^{(i)}$ has the right expectation for $\boldsymbol{x}^{(i)} = [x_1, \ldots, x_i, 0, \ldots, 0]$. Concretely, $\sum_{\boldsymbol{h} \in \mathcal{H}} P_{\mathcal{D}^{(i)}}[\boldsymbol{h}] \cdot \boldsymbol{h} = M\, \boldsymbol{x}^{(i)} + \boldsymbol{c}$. The desired result is obtained by setting $i = m$ as $\boldsymbol{v} = M\, \boldsymbol{x} + \boldsymbol{c}$ (see Appendix E). The base case $i = 0$ (i.e. before the first loop of the algorithm) is indeed true, since $\mathcal{D}^{(0)}$ is initialized to follow $P_{\mathcal{D}^{(0)}}[\boldsymbol{c}] = 1$, and $\boldsymbol{x}^{(0)} = \boldsymbol{0}$, thus we have $\boldsymbol{v}^{(0)} = M\boldsymbol{x}^{(0)} + \boldsymbol{c} = \boldsymbol{c}$. Now assume that by the end of the $(k-1)$st iteration we have the right distribution $\mathcal{D}^{(k-1)}$, namely $\boldsymbol{v}^{(k-1)} = \sum_{\boldsymbol{h} \in \mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h}$. Also assume that the kth comparator is applied on ith and jth element[14]. Thus the kth column of M will be $M_k = \boldsymbol{e}_i - \boldsymbol{e}_j$. Now, according to (4) the swap capacity at kth comparator is:

$$\begin{aligned} x_k + s_k &= v_j^{k-1} - v_i^{k-1} \\ &= \sum_{\boldsymbol{h} \in \mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot (h_j - h_i) \\ &= -\sum_{\boldsymbol{h} \in \mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot M_k^T\, \boldsymbol{h} \\ &= -M_k^T \sum_{\boldsymbol{h} \in \mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \end{aligned} \tag{6}$$

14. Note that $j > i$ as in sorting networks the swap value is propagated to lower wires

Now observe:

$$
\begin{aligned}
\boldsymbol{v}^{(k)} &= M\boldsymbol{x}^{(k)} + \boldsymbol{c} \\
&= x_k M_k + M\boldsymbol{x}^{(k-1)} + \boldsymbol{c} \\
&= x_k M_k + \boldsymbol{v}^{(k-1)} \\
&= x_k M_k + \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \\
&= \frac{x_k}{x_k + s_k} M_k \left(x_k + s_k\right) + \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \\
&= -\frac{x_k}{x_k + s_k} M_k M_k^T \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} + \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \qquad \text{According to (6)} \\
&= \left(I - \frac{x_k}{x_k + s_k} M_k M_k^T\right) \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \\
&= \left(\frac{s_k}{x_k + s_k} I + \frac{x_k}{x_k + s_k} T_{ij}\right) \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}] \cdot \boldsymbol{h} \\
&= \sum_{\boldsymbol{h}\in\mathcal{H}} \underbrace{\frac{s_k}{x_k + s_k} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}]}_{P_{\mathcal{D}^{(k)}}[\boldsymbol{h}]} \cdot \boldsymbol{h} + \underbrace{\frac{x_k}{x_k + s_k} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}]}_{P_{\mathcal{D}^{(k)}}[T_{ij}\,\boldsymbol{h}]} \cdot T_{ij}\,\boldsymbol{h} \\
&= \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k)}}[\boldsymbol{h}] \cdot \boldsymbol{h}
\end{aligned}
$$

in which T_{ij} is the row-switching matrix obtained by swapping the ith and jth rows of the identity matrix. For Huffman trees, the linear maps introduced in Pashkovich (2012) are used to set the depths of the leaves. It is straightforward to see that these linear maps maintain the equality $\boldsymbol{v}^{(k)} = \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k)}}[\boldsymbol{h}] \cdot \boldsymbol{h}$ when applied to $\boldsymbol{v}^{(k)}$ and all $\boldsymbol{h}$'s in $\mathcal{H}$. This concludes the inductive proof.

The final distribution $\mathcal{D}$ over objects $\boldsymbol{h} \in \mathcal{H}$ is decomposed into individual actions of swap/pass through the network of comparators independently. Thus one can draw an instance according to the distribution by simply doing independent Bernoulli trials associated with the comparators. It is also easy to see that the time complexity of the algorithm is $\mathcal{O}(n + m)$ since one just needs to do m Bernoulli trials. ■

Appendix H. Additional Loss with Approximate Projection

Each iteration of Bregman Projection is described in Appendix F. Since it is basically finding a positive root of a polynomial (which $n/(n+m)$ of the time is quadratic), each iteration is arguably efficient. Now suppose, using iterative Bregman projections, we reached at $\widehat{\boldsymbol{w}} = (\widehat{\boldsymbol{v}}, \widehat{\boldsymbol{x}}, \widehat{\boldsymbol{s}})$ which is ϵ-close to the exact projection $\boldsymbol{w} = (\boldsymbol{v}, \boldsymbol{x}, \boldsymbol{s})$, that is $\|\boldsymbol{w} - \widehat{\boldsymbol{w}}\|_2 < \epsilon$. In this analysis, we work with a two-level approximation: 1) approximating mean vector $\boldsymbol{v}$ by the mean vector $\widetilde{\boldsymbol{v}} := M\widehat{\boldsymbol{x}} + \boldsymbol{c}$ and 2) approximating the mean vector $\widetilde{\boldsymbol{v}}$ by the mean vector $\boldsymbol{v}(\widehat{\boldsymbol{p}})$ (where $\widehat{\boldsymbol{p}} = \widehat{\boldsymbol{x}}/(\widehat{\boldsymbol{x}} + \widehat{\boldsymbol{s}})$ with coordinate-wise division) obtained from Algorithm 3 with $\widehat{\boldsymbol{x}}$ and $\widehat{\boldsymbol{s}}$ as input. First, observe that:

$$\begin{aligned}\|\boldsymbol{v}-\widetilde{\boldsymbol{v}}\|_2 &= \|M\,(\boldsymbol{x}-\widehat{\boldsymbol{x}})\|_2\\ &\leq \|M\|_F\,\|\boldsymbol{x}-\widehat{\boldsymbol{x}}\|_2\\ &\leq (\sqrt{2\,n})\,\epsilon \end{aligned} \tag{7}$$

Now suppose we run Algorithm 3 with $\widehat{\boldsymbol{x}}$ and $\widehat{\boldsymbol{s}}$ as input. Similar to Appendix G, let M_k be the k-th column of M, and let $T_{\alpha\beta}$ be the row-switching matrix that is obtained from switching α-th and β-th row in identity matrix. Additionally, let $\boldsymbol{v}^{(k)}(\widehat{\boldsymbol{p}})$ be the mean vector associated with the distribution $\mathcal{D}^{(k)}$ obtained by the end of k-th loop of the Algorithm 3 i.e. $\boldsymbol{v}^{(k)}(\widehat{\boldsymbol{p}}) := \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k)}}[\boldsymbol{h}]\cdot\boldsymbol{h}$ (so $\boldsymbol{v}^{(m)}(\widehat{\boldsymbol{p}}) = \boldsymbol{v}(\widehat{\boldsymbol{p}})$). Also for all $k\in\{1..m\}$ define $\widetilde{\boldsymbol{v}}^{(k)} := \boldsymbol{c}+\sum_{i=1}^{k} M_i\,\widehat{x}_i$ (thus $\widetilde{\boldsymbol{v}}^{(m)} = \widetilde{\boldsymbol{v}}$). Furthermore, let $\boldsymbol{\delta}^{(k)} := \boldsymbol{v}^{(k)}(\widehat{\boldsymbol{p}}) - \widetilde{\boldsymbol{v}}^{(k)}$. Now we can write:

$$\begin{aligned}\boldsymbol{v}^{(k)}(\widehat{\boldsymbol{p}}) &= \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k)}}[\boldsymbol{h}]\cdot\boldsymbol{h}\\
&= \sum_{\boldsymbol{h}\in\mathcal{H}} \frac{\widehat{s}_k}{\widehat{x}_k+\widehat{s}_k} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}]\cdot\boldsymbol{h} + \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}]\cdot T_{\alpha\beta}\,\boldsymbol{h}\\
&= (\frac{\widehat{s}_k}{\widehat{x}_k+\widehat{s}_k}\,I + \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,T_{\alpha\beta}) \sum_{\boldsymbol{h}\in\mathcal{H}} P_{\mathcal{D}^{(k-1)}}[\boldsymbol{h}]\cdot\boldsymbol{h}\\
&= (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\boldsymbol{v}^{(k-1)}(\widehat{\boldsymbol{p}}) \qquad \text{since } I - T_{\alpha\beta} = M_k\,M_k^T\\
&= (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\widetilde{\boldsymbol{v}}^{(k-1)} + (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)}\\
&= (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\left(\boldsymbol{c}+\sum_{i=1}^{k-1} M_i\,\widehat{x}_i\right) + (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)}\\
&= \left(\boldsymbol{c}+\sum_{i=1}^{k-1} M_i\,\widehat{x}_i\right) - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T\,\left(\boldsymbol{c}+\sum_{i=1}^{k-1} M_i\,\widehat{x}_i\right)\\
&\qquad + (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)}\\
&= \widetilde{\boldsymbol{v}}^{(k)} \underbrace{-M_k\widehat{x}_k - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T\,\left(\boldsymbol{c}+\sum_{i=1}^{k-1} M_i\,\widehat{x}_i\right) + (I - \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)}}_{\boldsymbol{\delta}^{(k)}}\end{aligned}$$

Now define $\widehat{p}_k := \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}$ and let $\text{err}_k := -\,M_k^T\,\left(\boldsymbol{c}+\sum_{i=1}^{k-1} M_i\,\widehat{x}_i\right) - (\widehat{x}_k+\widehat{s}_k)$, which is – according to Lemma 6 – the error in the k-th row of $A\boldsymbol{x}+\boldsymbol{s}=\boldsymbol{b}$ using $\widehat{\boldsymbol{x}}$ and $\widehat{\boldsymbol{s}}$ i.e. amount by which $(A\widehat{\boldsymbol{x}}+\widehat{\boldsymbol{s}})_k$ falls short of b_k, violating the k-th constraint of $A\boldsymbol{x}+\boldsymbol{s}=b$. Thus we obtain:

$$\begin{aligned}\boldsymbol{\delta}^{(k)} &= -M_k\widehat{x}_k + \widehat{p}_k\,M_k\,(\widehat{x}_k+\widehat{s}_k+\text{err}_k) + (I-\widehat{p}_k\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)}\\
&= \widehat{p}_k\,M_k\,\text{err}_k + (I-\widehat{p}_k\,M_k\,M_k^T)\,\boldsymbol{\delta}^{(k-1)} \qquad\qquad \widehat{p}_k = \frac{\widehat{x}_k}{\widehat{x}_k+\widehat{s}_k}\end{aligned}$$

Observe that $\boldsymbol{\delta}^{(0)} = \boldsymbol{c} - \boldsymbol{c} = \mathbf{0}$. Thus, by unrolling the recurrence relation above, we have:

$$\boldsymbol{v}(\widehat{\boldsymbol{p}}) - \widetilde{\boldsymbol{v}} = \boldsymbol{\delta}^{(m)} = \sum_{k=1}^{m} \widehat{p}_k \, M_k \, \mathrm{err}_k \prod_{i=k+1}^{m} (I - \widehat{p}_i \, M_i \, M_i^T)$$

Note that since $I - \widehat{p}_i \, M_i \, M_i^T$ is a $n \times n$ doubly-stochastic matrix, $\prod_{i=1}^{k-1}(I - \widehat{p}_i \, M_i \, M_i^T)$ is also a $n \times n$ doubly-stochastic matrix, and consequently, its Frobenius norm is at most $\sqrt{n}$. Thus we have:

$$\begin{aligned}
\|\boldsymbol{v}(\widehat{\boldsymbol{p}}) - \widetilde{\boldsymbol{v}}\|_2 = \|\boldsymbol{\delta}^{(m)}\|_2 &\leq \sum_{k=1}^{m} |\widehat{p}_k| \, \|M_k\|_2 \, |\mathrm{err}_k| \, \sqrt{n} \leq \sqrt{2\,n} \sum_{k=1}^{m} |\mathrm{err}_k| \\
&= \sqrt{2\,n} \, \|\mathbf{err}\|_1 && \mathbf{err} := (\mathrm{err}_1, \ldots, \mathrm{err}_m) \\
&\leq \sqrt{2\,n\,m} \, \|\mathbf{err}\|_2 && (8)
\end{aligned}$$

Observe that we can bound the 2-norm of the vector **err** as follows:

$$\begin{aligned}
\|\mathbf{err}\|_2 &= \| - A\widehat{\boldsymbol{x}} - \widehat{\boldsymbol{s}} + \boldsymbol{b}\|_2 \\
&= \|A(\boldsymbol{x} - \widehat{\boldsymbol{x}}) + (\boldsymbol{s} - \widehat{\boldsymbol{s}})\|_2 && \boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{s} \\
&\leq \|A\|_F \, \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_2 + \|\boldsymbol{s} - \widehat{\boldsymbol{s}}\|_2 \\
&\leq \|M^T M + I\|_F \, \epsilon + \epsilon \\
&\leq \left(\|M^T\|_2 \, \|M\|_2 + \|I\|_2\right) \, \epsilon + \epsilon \\
&= (2n + \sqrt{n} + 1) \, \epsilon && (9)
\end{aligned}$$

Therefore, if we perform Algorithm 3 with inputs $\widehat{\boldsymbol{x}}$ and $\widehat{\boldsymbol{s}}$, combining the inequalities (7), (8), and (9), the generated mean vector $\boldsymbol{v}(\widehat{\boldsymbol{p}})$ can be shown to be close to the mean vector $\boldsymbol{v}$ associated with the exact projection:

$$\begin{aligned}
\|\boldsymbol{v} - \boldsymbol{v}(\widehat{\boldsymbol{p}})\|_2 &\leq \|\boldsymbol{v} - \widetilde{\boldsymbol{v}}\|_2 + \|\widetilde{\boldsymbol{v}} - \boldsymbol{v}(\widehat{\boldsymbol{p}})\|_2 \\
&\leq (\sqrt{2\,n}) \, \epsilon + \sqrt{2\,n\,m} \, (2n + \sqrt{n} + 1) \, \epsilon \\
&= \sqrt{2\,n} \, (1 + \sqrt{m} \, (2n + \sqrt{n} + 1)) \, \epsilon
\end{aligned}$$

Now we can compute the total expected loss using approximate projection:

$$\begin{aligned}
\left| \sum_{t=1}^{T} \boldsymbol{v}^{t-1}(\widehat{\boldsymbol{p}}) \cdot \boldsymbol{\ell}^t \right| &= \left| \sum_{t=1}^{T} \left(\boldsymbol{v}^{t-1} + (\boldsymbol{v}^{t-1} - \boldsymbol{v}^{t-1}(\widehat{\boldsymbol{p}}))\right) \cdot \boldsymbol{\ell}^t \right| \\
&= \left| \sum_{t=1}^{T} \boldsymbol{v}^{t-1} \cdot \boldsymbol{\ell}^t + \sum_{t=1}^{T} (\boldsymbol{v}^{t-1} - \boldsymbol{v}^{t-1}(\widehat{\boldsymbol{p}})) \cdot \boldsymbol{\ell}^t \right| \\
&\leq \left| \sum_{t=1}^{T} \boldsymbol{v}^{t-1} \cdot \boldsymbol{\ell}^t \right| + \left| \sum_{t=1}^{T} (\boldsymbol{v}^{t-1} - \boldsymbol{v}^{t-1}(\widehat{\boldsymbol{p}})) \cdot \boldsymbol{\ell}^t \right| \\
&\leq \left| \sum_{t=1}^{T} \boldsymbol{v}^{t-1} \cdot \boldsymbol{\ell}^t \right| + \sum_{t=1}^{T} \|\boldsymbol{v}^{t-1} - \boldsymbol{v}^{t-1}(\widehat{\boldsymbol{p}})\|_2 \, \|\boldsymbol{\ell}^t\|_2 \\
&\leq \left| \sum_{t=1}^{T} \boldsymbol{v}^{t-1} \cdot \boldsymbol{\ell}^t \right| + T \, \left(\sqrt{2\,n} \, (1 + \sqrt{m} \, (2n + \sqrt{n} + 1)) \, \epsilon\right) \, \sqrt{n}
\end{aligned}$$

For Huffman trees, the linear maps introduced in Pashkovich (2012) have this property that $\|F(\boldsymbol{a}) - F(\boldsymbol{a}')\|_2 \leq \|\boldsymbol{a} - \boldsymbol{a}'\|_2$ for all vectors $\boldsymbol{a}$ and $\boldsymbol{a}'$ where $F(\cdot)$ is the linear map. Using this property, it is straightforward to observe that this analysis can be extended for Huffman trees in which these linear maps are also used along with the reflection relations.

Setting $\epsilon = \frac{1}{(\sqrt{2}\,n)\,(1+\sqrt{m}\,(2n+\sqrt{n}+1))\,T}$, we add at most one unit to the expected cumulative loss with exact projections.