
Online Dynamic Programming

Holakou Rahmanian

Department of Computer Science
University of California Santa Cruz
Santa Cruz, CA 95060
holakou@ucsc.edu

S.V.N. Vishwanathan

Department of Computer Science
University of California Santa Cruz
Santa Cruz, CA 95060
vishy@ucsc.edu

Manfred K. Warmuth

Department of Computer Science
University of California Santa Cruz
Santa Cruz, CA 95060
manfred@ucsc.edu

Abstract

We consider the problem of repeatedly solving a variant of the same dynamic programming problem in successive trials. An instance of the type of problems we consider is to find the optimal binary search tree. At the beginning of each trial, the learner probabilistically chooses a tree with the n keys at the internal nodes and the $n + 1$ gaps between keys at the leaves. It is then told the frequencies of the keys and gaps and is charged by the average search cost for the chosen tree. The problem is online because the frequencies can change between trials. The goal is to develop algorithms with the property that their total average search cost (loss) in all trials is close to the total loss of the best tree chosen in hind sight for all trials. The challenge, of course, is that the algorithm has to deal with exponential number of trees. We develop a methodology for tackling such problems for a wide class of dynamic programming algorithms. Our framework allows us to extend online learning algorithms like *Hedge* [9] and *Component Hedge* [15] to a significantly wider class of combinatorial objects than was possible before.

1 Introduction

Consider the following *online learning* game. In each round, the algorithm plays with a Binary Search Tree (BST) for a given set of n keys. Then the adversary reveals a set of probabilities for the n keys and their $n + 1$ gaps, and the algorithm incurs a loss of *average search cost*. The goal is to predict with a sequence of BSTs minimizing *regret* which is the difference between the total loss of the algorithm and that of the single best BST chosen in hindsight.

A natural approach to solve this problem is to keep track of a distribution on all possible BSTs over the trials of the game. However, if implemented naively, this approach will be impractical since it is intractable to maintain a weight vector of exponential size. Going beyond BSTs, this issue is common among all combinatorial objects with n components as the number of objects is typically exponential in n .

There has been much work on developing efficient algorithms that implicitly encode the weights over the set of combinatorial objects using concise representations. Such work includes k -sets [22], permutations [1, 12, 23] and paths [3, 7, 11, 16, 21]. There are also more general tools for learning combinatorial concepts. *Follow the Perturbed Leader (FPL)* [13] is a simple algorithm which adds random perturbation to the cumulative loss of each component, and then predicts with the combinatorial object with minimum perturbed loss. *Component Hedge (CH)* [15] and its extensions

to submodular base polytopes [10, 19, 20] are powerful generic techniques which keep track of component-wise weights in the convex hull of the objects for prediction. Unfortunately the results of CH and its current extensions cannot be directly applied to problems like BST. This is because there is no polynomial characterization of the convex hull of BSTs in its original space.

We close this gap by exploiting the underlying dynamic programming algorithm which solves the BST optimization problem. Each BST problem is connected to two smaller BST problems once the root is decided. We “hedge” our bets on these localized decisions. Our method is general and applies to combinatorial objects whose optimization problem can be solved efficiently via a wide class of dynamic programming algorithms. Using the underlying graph of sub-problems in dynamic programming, we form a concise representation of the combinatorial objects by encoding them as sub-graphs. This graph representation allows us to extend both standard Hedge algorithm [9, 17] and Component Hedge to wide class of combinatorial objects like BST (see Section 5), Matrix-Chain Multiplication, Knapsack, Rod Cutting, and Weighted Interval Scheduling (see Appendix A).

Paper Outline We start with online learning of paths as simple sub-graphs in Section 2. This section briefly describes the main two existing algorithms for the path problem: (1) *path kernels* – which is an efficient implementation of Hedge – and (2) Component Hedge. Section 3 introduces a new class of sub-graphs, namely *k-multipaths*, and generalizes algorithms in Section 2 for these combinatorial objects. In Section 4, we precisely define a class of combinatorial objects recognized by dynamic programming algorithms and their corresponding online prediction problem – namely *dynamic programming games*. Then we prove that these dynamic programming games reduce to online learning of *k-multipaths*. As an interesting example, we apply our method to the problem of online learning of binary search trees in Section 5 (due to space restriction, additional examples are discussed in Appendix A). Finally, Section 6 concludes with comparison to other algorithms and future work.

2 Background

Perhaps the simplest algorithms in online learning are the well-known so-call “experts algorithms” like the Randomized Weighted Majority [17] or Hedge [9] algorithms. It keeps track of a probability vector over all experts and the weight w_i of expert i is proportional to $\exp(-\eta L(i))$, where $L(i)$ is the cumulative loss of expert i through trials and η is a non-negative learning rate. In our application, we use the typically exponentially many combinatorial objects as the set of experts. Like [15], in order to make a clear distinction with Component Hedge, from now on in this paper we call this algorithm *Expanded Hedge (EH)*.

2.1 Learning Paths

The online shortest path has been explored both in full information setting [15, 21] and various bandit settings [2, 3, 8, 11]. Concretely the problem in full information setting is as follows. Consider a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$ with designated source node $s \in V$ and sink node $\omega \in V$. In each trial, the algorithm predicts with a path from s to ω . Then, for each edge $e \in E$, the adversary reveals a loss $\ell_e \in [0, 1]$. The loss of the algorithm is given by the sum of the losses of the edges along the predicted path. The goal is to minimize the regret which is the difference between the total loss of the algorithm and that of the single best path chosen in hindsight. There are two main algorithms in the literature for this setting:

2.1.1 Expanded Hedge on Paths

Takimoto and Warmuth [21] developed an algorithmic approach, namely *path kernels*, to apply Expanded Hedge on path problem efficiently by exploiting the underlying structure of the paths. Basically, path kernels maintain a distribution of the paths by keeping the weights on the edges along the paths normalized. In each trial, the weight of each edge is updated multiplicatively by its associated exponentiated loss followed by normalization via *weight pushing* [18]. Finally, to sample from the distribution, starting from the source, we hop to the next node according to localized distribution over the out-going edges at each node. The regret bound is similar to the one of the Hedge algorithm [9]:

Theorem 1 (Takimoto-Warmuth [21]). *Given a DAG $\mathcal{G} = (V, E)$ with designated source node $s \in V$ and sink node $\omega \in V$, let \mathcal{N} and L^* be the number of paths in \mathcal{G} from s to ω and the total loss of best path, respectively. Also let B be an upper bound on the loss of any path in each trial. Then, over T trials, EH guarantees:*

$$\mathbb{E}[L_{EH}] - L^* \leq B \sqrt{2T \log \mathcal{N}} + B \log \mathcal{N}$$

2.1.2 Component Hedge on Paths

Koolen, Warmuth and Kivinen [15] developed a generic framework called *Component Hedge (CH)* which results in efficient and effective online algorithm over combinatorial objects. When applied to the path problem, CH maintains a “usage” vector in a *unit-flow polytope*. In each trial, the weight of each component (i.e. edge) is updated multiplicatively by its associated exponentiated loss. Then the weight vector is projected back to the unit-flow polytope via relative entropy projection. To do projection, iterative Bregman projection [5] is used which basically enforces flow conservation at each node by setting input and output flow to their geometric average as it cycles through the vertices. Finally, to sample with the same expectation as the usage vector, the usage vector is decomposed into paths using a greedy approach which zeros out at least one edge in each iteration. The regret bound will no longer have an extra range factor:

Theorem 2 (Koolen-Warmuth-Kivinen [15]). *Given a DAG $\mathcal{G} = (V, E)$ with designated source node $s \in V$ and sink node $\omega \in V$, let D be the length the paths in \mathcal{G} from s to ω against which the CH algorithm is compared. Also denote the total loss of the best path of length D by L^* . Then, over T trials, CH guarantees:*

$$\mathbb{E}[L_{CH}] - L^* \leq D \sqrt{4T \log |V|} + 2D \log |V|$$

3 Learning k -Multipaths

We now generalize the online shortest path problem from learning paths to learning more complex subgraphs – namely *k -multipaths*. To do so, we also assume some additional structure in the underlying DAG which we call *k -regularity*.

Definition 1 (k -Regular DAG). *A DAG $\mathcal{G} = (V, E)$ is called **k -regular** if and only if it has following properties:*

- (i) *There exists one designated “source” node $s \in V$ with no in-coming edges.*
- (ii) *There exists a set of “sink” nodes $\Omega \subset V$ which is the set of nodes with no out-going edges.*
- (iii) *For each vertex $v \in V - \Omega$, the set of out-going edges from v (denoted by E_v) is partitioned into tuples of size k (or simply **k -tuples**) denoted by $\mathcal{F}_v = \{E_v^{(1)}, \dots, E_v^{(d_v)}\}$ in which¹*

$$\bigcup_{i \in [d_v]} E_v^{(i)} = E_v, \quad \forall i, j \in [d_v], i \neq j, E_v^{(i)} \cap E_v^{(j)} = \emptyset, \quad \forall i \in [d_v], |E_v^{(i)}| = k$$

Definition 2 (k -Multipath). *Given a k -regular DAG $\mathcal{G} = (V, E)$, let² $\pi \in \mathbb{N}^{|E|}$ in which π_e is associated with $e \in E$. Define $\pi_{in}(v) := \sum_{e: e=(u,v)} \pi_e$ and $\pi_{out}(v) := \sum_{e: e=(v,u)} \pi_e$. π is called a **k -multipath** if and only if it has the properties below:*

- (i) $\pi_{out}(s) = k$.
- (ii) *For any two edges e and e' belonging to the same k -tuple in \mathcal{G} , $\pi_e = \pi_{e'}$.*
- (iii) *For each vertex $v \in V - \Omega - \{s\}$, $\pi_{out}(v) = k \times \pi_{in}(v)$.*

Intuitively, k -multipath is a more general version of k -ary trees in the sense that as k children are branching out of a given parent, some children may have already been explored from other branches. Similar to nodes, an edge can also visited more than once in a k -multipath. See Figure 1 for an example of 2-multipath in given 2-regular DAG.

¹For a positive integer j , $[j] := \{1, 2, \dots, j\}$

² \mathbb{N} is the set of non-negative integers.

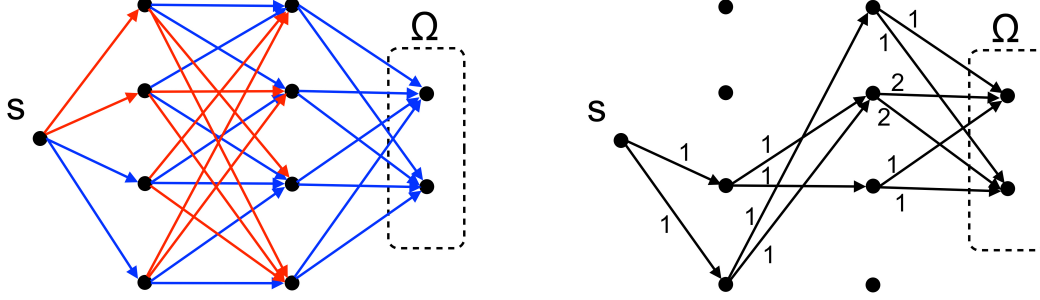


Figure 1: (left) An example of 2-regular DAG. The partition of out-going edges at each node is depicted by different colors of red and blue. (right) An example of 2-multipath in the 2-regular DAG.

k -Multipath Learning Problem Now let us define the problem of *online learning of k -multipaths*. Consider a k -regular DAG \mathcal{G} with parameters as in Definition 1. At trial t , the algorithm predicts with a k -multipath $\pi^{(t-1)}$. Then, for each edge $e \in E$, the adversary reveals a loss $\ell_e^{(t)} \in [0, 1]$. The loss of the algorithm is given by $\pi^{(t-1)} \cdot \ell^{(t)}$. Observe that the online shortest path problem is a special case when $|\Omega| = k = 1$. In the remainder of this section, we generalize the algorithms in 2.1.1 and 2.1.2 for online learning of k -multipaths.

3.1 Expanded Hedge on k -Multipaths

We implement the Expanded Hedge efficiently over the problem of online learning of k -multipath by considering each k -multipath as an expert. Notice that each k -multipath can be generated by successively choosing a k -tuple out of out-going edges from a given node $v \in V$ by starting from the source s until we reach sink nodes in Ω . With **each k -tuple** in the k -regular DAG, i.e. $\tilde{E} \in \mathcal{F} := \bigcup_{v \in V} \mathcal{F}_v$, we associate a weight $w_{\tilde{E}}$. We maintain a distribution W over k -multipaths by keeping the weights $w \in \mathbb{R}_+^{|\mathcal{F}|}$ along the k -tuples in k -multipaths normalized. Concretely the weight vector w has the properties below which is denoted by Φ throughout this paper:

- Φ_1 : The weights are in *product form*, i.e. $W_\pi = \prod_{\tilde{E} \in \mathcal{F}} (w_{\tilde{E}})^{\pi_{\tilde{E}}}$ in which $\pi_{\tilde{E}}$ is the common value in π among edges $e \in \tilde{E}$.
- Φ_2 : The weights are *locally normalized*, i.e. $\sum_{\tilde{E} \in \mathcal{F}_v} w_{\tilde{E}} = 1$ for all $v \in V - \Omega$.
- Φ_3 : The total path weight is one, i.e. $\sum_{\pi} W_\pi = 1$.

If W has the Φ properties, one can sample from W as follows. Starting from the source $v = s$, sample an outgoing k -tuple from \mathcal{F}_v as the weights form a normalized distribution (See Φ_2). Continue sampling k -tuples until the k -multipath is “fully grown” and reaches sink nodes in Ω . Because of Φ_1 , this is a valid sample drawn from W . Observe that $\pi_{\tilde{E}}$ indicates the number of times k -tuple \tilde{E} is sampled through this process. Assuming all k -multipaths π satisfy $\|\pi\|_1 \leq \bar{D}$, this can be done in $\mathcal{O}(\frac{\bar{D}}{k} \times \frac{|V|}{k})$. The updates for each k -multipath π according to Expanded Hedge are as follows:

$$\begin{aligned}
 W_\pi^{(t)} &\propto W_\pi^{(t-1)} \exp(-\eta \pi \cdot \ell^{(t)}) \\
 &= \left(\prod_{\tilde{E} \in \mathcal{F}} (w_{\tilde{E}}^{(t-1)})^{\pi_{\tilde{E}}} \right) \exp \left[-\eta \sum_{\tilde{E} \in \mathcal{F}} \pi_{\tilde{E}} \left(\sum_{e \in \tilde{E}} \ell_e^{(t)} \right) \right] = \prod_{\tilde{E} \in \mathcal{F}} \left(\underbrace{w_{\tilde{E}}^{(t-1)} \exp \left[-\eta \sum_{e \in \tilde{E}} \ell_e^{(t)} \right]}_{\hat{w}_{\tilde{E}}^{(t)}} \right)^{\pi_{\tilde{E}}}
 \end{aligned}$$

which basically indicates that in order to implement Expanded Hedge, the weight of each k -tuple $\tilde{E} \in \mathcal{F}$ can be updated multiplicatively with exponentiated loss associated with \tilde{E} , i.e. $\hat{w}_{\tilde{E}}^{(t)} = w_{\tilde{E}}^{(t-1)} \exp \left[-\eta \sum_{e \in \tilde{E}} \ell_e^{(t)} \right]$. Now we explore the efficient normalization of $W^{(t)}$.

Generalized Weight Pushing For efficient normalization, we generalize the *weight pushing algorithm* [18] for k -multipaths. Our goal is to find new k -tuple weights $w_{\tilde{E}}^{(t)}$ such that Φ properties are satisfied and $W_{\pi}^{(t)} \propto \prod_{\tilde{E} \in \mathcal{F}} (\hat{w}_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}}$. To do so, for each $v \in V$, we introduce Z_v as the normalization constant if v was the source in \mathcal{G} . Now the *generalized weight pushing* is as follows:

1. For every $v \in \Omega$, set $Z_v = 1$.
2. Recursively find Z_v for all $v \in V - \Omega$ via $Z_v = \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \prod_{u:(v,u) \in \tilde{E}} Z_u$.
3. For each $v \in V - \Omega$, for all $\tilde{E} \in \mathcal{F}_v$, set $w_{\tilde{E}}^{(t)} = \hat{w}_{\tilde{E}}^{(t)} \frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v}$

See Appendix B for the proof of correctness and efficiency of the generalized weight pushing algorithm.

Regret Bound In order to use the regret bound of Expanded Hedge [9], we have to initialize $W^{(0)}$ to the uniform distribution. This can be done by setting all weights to one followed by generalized weight pushing. Note that Theorem 1 is a corollary of Theorem 3 if we set $k = 1$.

Theorem 3. *Given a k -regular DAG \mathcal{G} with designated source node s and sink nodes Ω , let \mathcal{N} and L^* be the number of k -multipaths in \mathcal{G} from s to Ω and the total loss of best k -multipath, respectively. Also let B be an upper bound on the loss of any k -multipath in each trial. Then, over T trials, EH guarantees*

$$\mathbb{E}[L_{EH}] - L^* \leq B \sqrt{2T \log \mathcal{N}} + B \log \mathcal{N}$$

3.2 Component Hedge on k -Multipaths

In order to apply Component Hedge to the online learning of k -multipath, we associated a weight w_e with **each edge** $e \in E$. As briefly introduced in Section 2.1.2, CH consists of different modules. We discuss the modules of this algorithm introduced in [15] followed by the regret bound guarantees.

k -Flow Polytope The polytope of k -multipaths (denoted by Ψ) can be obtained from Definition 2 by relaxing the integer constraint. For a given point $\mathbf{w} \in \mathbb{R}_+^{|E|}$, define $w_{\text{in}}(v) := \sum_{e:e=(u,v)} w_e$ and $w_{\text{out}}(v) := \sum_{e:e=(v,u)} w_e$. The constraints below on \mathbf{w} characterizes the polytope Ψ :

1. **Root Outflow** $w_{\text{out}}(s) = k$.
2. **k -Flow Preservation** For each vertex $v \in V - \Omega - \{s\}$, $w_{\text{out}}(v) = k \times w_{\text{in}}(v)$.
3. **k -Tuple Equality** For any two edges e and e' belonging to the same k -tuple in \mathcal{G} , $w_e = w_{e'}$.

Relative Entropy Projection In t th trial, after the multiplicative update of the weight vector by exponentiated loss (i.e. $\forall e \in E$, $\hat{w}_e^{(t)} = w^{(t-1)} e^{-\eta \ell_e^{(t)}}$), the weight vector is projected back to the polytope Ψ via relative entropy projection. Concretely:

$$\mathbf{w}^{(t)} := \arg \min_{\mathbf{w} \in \Psi} \Delta(\mathbf{w} || \hat{\mathbf{w}}^{(t)}), \quad \text{where} \quad \Delta(\mathbf{a} || \mathbf{b}) = \sum_i a_i \log \frac{a_i}{b_i} + b_i - a_i$$

To do this projection, iterative Bregman projection [5] can be used which cycles through the constraints and project the point to each constraint iteratively. The details of the relative entropy projection to each constraint is shown in Appendix C. These projections have the following algebraic interpretations:

1. **Root Outflow** Normalize the out-flow of s to k .
2. **k -Flow Preservation** For a given vertex $v \in V - \Omega - \{s\}$ multiplicatively scale up/down the weights associated with in- and out-flow so that they will be proportionate to the k -to-1 weighted geometric average of the out- and in-flow, respectively.
3. **k -Tuple Equality** Enforce equality by assigning the geometric average of each k -tuple to its components. This can be done simultaneously to all k -tuples in \mathcal{F} .

Decomposition First we find any k -multipath with non-zero weights on all edges. To do so, we keep choosing an out-going non-zero k -tuple from a given node v , starting from $v = s$, until we reach the sink nodes in Ω . Assuming all k -multipaths π satisfy $\|\pi\|_1 \leq \bar{D}$, this can be done in $\mathcal{O}(\frac{\bar{D}}{k} \times \frac{|V|}{k})$. Then we subtract that k -multipath, scaled by its minimum edge weight. This creates a new zero, maintains k -flow preservation and k -tuple equality constraints, and reduces the source's outflow. After at most $\mathcal{O}(\frac{|V|^2}{k})$ iterations the source has outflow zero. The total running time is $\mathcal{O}(\frac{\bar{D}|V|^3}{k^3})$.

Regret Bound The general regret bound for Component Hedge depends on the initial weight vector $w^{(0)} \in \Psi$ via $\Delta(\pi||w^{(0)})$ in which π is the k -multipath the algorithm is compared against. Instead of explicitly selecting $w^{(0)} \in \Psi$, we implicitly design the initial point by starting with a point $\tilde{w}^{(0)} \in \mathbb{R}_+^{|E|}$ and project it onto Ψ . This novel idea yields to regret guarantee below (see Appendix D for proof):

Theorem 4. *Given a k -regular DAG $\mathcal{G} = (V, E)$, let D be the upperbound for the 1-norm of the k -multipaths in \mathcal{G} against which the algorithm is compared³. Also denote the total loss of the best k -multipath by L^* . Assuming $D \leq |E|$, then, over T trials, CH guarantees*

$$\mathbb{E}[L_{CH}] - L^* \leq D \sqrt{8T \log |V|} + 4D \log |V| \quad (1)$$

Moreover, if the infinity-norm of the k -multipaths is one (i.e. π is a bit-vector), then:

$$\mathbb{E}[L_{CH}] - L^* \leq D \sqrt{4T \log |V|} + 2D \log |V| \quad (2)$$

Notice that by setting $|\Omega| = k = 1$, all modules of path learning in [15] are recovered. Moreover, observe that Theorem 2 is a corollary of Theorem 4 since every path is represented as a bit vector.

4 Dynamic Programming Games

In this section, we focus on the online learning of combinatorial objects which can be “recognized” by a dynamic programming algorithm – *dynamic programming games*. First we precisely define the problem and then show the its connection to k -multipath learning problem

Problem Description Let \mathcal{P} be a minimization⁴ problem in \mathbb{R}^n which can be solved efficiently via dynamic programming. The solution is a combinatorial object minimizing a given loss function. Denote V as the set of all subproblems v in dynamic programming which needs to be solved in order to eventually solve \mathcal{P} iteratively in bottom-up fashion [6]. Also let $\Omega \subset V$ be the set of base/primitive subproblems. Assuming $u, v \in V$, we focus on the family of dynamic programming algorithm with the recurrence relation of the form below:

$$\text{OPT}(v) = \begin{cases} L_\Omega(v) & v \in \Omega \\ \min_{U \in \mathcal{H}_v} [\sum_{u \in U} \text{OPT}(u) + L(v, U)] & v \in V - \Omega \end{cases} \quad (3)$$

such that these two conditions hold: (1) there exists $k \in \mathbb{N}$ such that $|U| = k$ for all v and $U \in \mathcal{H}_v$ and (2) for all v , all U 's in \mathcal{H}_v are mutually exclusive. Note that $L(v, U) : V \times 2^V \rightarrow \mathbb{R}_+$ and $L_\Omega(v) : V \rightarrow \mathbb{R}_+$ encode the given loss function into dynamic programming recurrence relation. Also observe that every object can be recovered by choosing the “right” sequence of U 's as we go through the recurrence relation.

Now construct the *online* version of problem \mathcal{P} – denoted by $\mathcal{P}_{\text{online}}$ – by allowing the loss function to be changing over trials $t = 1 \dots T$. Concretely, instead of $L(v, U)$ we use $L_t(v, U)$ to reflect the online-ness of the problem. In each trial, the online algorithm predicts (perhaps randomly) with a combinatorial object recognized by problem \mathcal{P} . Then the adversary reveals the structured loss vector which can be encoded to function $L_t(v, U)$ according to the underlying dynamic programming algorithm. The goal is to minimize *regret* over the trials $t = 1 \dots T$.

³ D is usually in terms of k and $|V|$.

⁴Later in Appendix A we will provide maximization examples which can be solved with the same approach.

Reduction to Online Learning of k -Multipaths The dependency relations among the subproblems in dynamic programming algorithm can be encoded as a k -regular DAG \mathcal{G} which can be constructed as follows. For each subproblem $v \in V$ create a vertex. In particular set the source node $s \in V$ and the set of sink nodes $\Omega \subset V$ to be the original problem (i.e. \mathcal{P}) and the set of base/primitive subproblems, respectively. For every $v \in V - \Omega$, for all $U \in \mathcal{H}_v$ and for all $u \in U$, draw an edge $e = (v, u)$ with loss $\ell_e = \frac{1}{k}(L_t(v, U) + \sum_{u \in U \cap \Omega} L_\Omega(u))$, that is, uniformly distribute the “transition” loss $L_t(v, U)$ and “end” loss $\sum_{u \in U \cap \Omega} L_\Omega(u)$ to all edges from v to the members of U . Finally, for each $v \in V - \Omega$, form the k -tuples out of the outgoing edges E_v according to \mathcal{H}_v . Concretely $\mathcal{F}_v := \{\tilde{E} \mid \exists U \in \mathcal{H}_v \text{ s.t. } \tilde{E} = \{(v, u) \mid u \in U\}\}$

Every combinatorial object is a k -multipath π sourced at s with sinks in Ω . Moreover, the loss of that object is $\pi \cdot \ell$. Now if the given loss function of the problem is in such way that $\ell_e \in [0, 1]$, we can apply both algorithms CH and EH and their corresponding regret guarantees in Theorems 3 and 4, respectively ⁵.

5 Online Learning of Binary Search Trees

As an instantiation, let us now apply our method on an interesting combinatorial object recognized by dynamic programming. See Appendix A for more examples. Consider the online version of *optimal binary search tree* problem [6]. Concretely, we are given a sequence of n distinct keys $K_1 < K_2 < \dots < K_n$ to build a binary search tree (BST). Moreover, we are also given $n + 1$ “dummy keys” D_0, \dots, D_n indicating *search failures* and for all $i \in [n]$ we have $D_{i-1} < K_i < D_i$. In each trial t , the algorithm predicts with a BST $\gamma^{(t)}$. Then the adversary reveals the a probability vector $\ell^{(t)} = (\mathbf{p}^{(t)}, \mathbf{q}^{(t)})$ with $\mathbf{p}^{(t)} \in [0, 1]^n$, $\mathbf{q}^{(t)} \in [0, 1]^{n+1}$ and $\sum_{i=1}^n p_i^{(t)} + \sum_{i=0}^n q_i^{(t)} = 1$. For each i , $p_i^{(t)}$ and $q_i^{(t)}$ are the *search probabilities* for K_i and D_i , respectively. The loss is defined as the expected cost of a search in the BST γ which is basically average depth ⁶ of the nodes in the BST:

$$\text{loss}_\gamma^{(t)} = \sum_{i=1}^n \text{depth}_\gamma(K_i) \cdot p_i^{(t)} + \sum_{j=0}^n \text{depth}_\gamma(D_j) \cdot q_j^{(t)}$$

5.1 Challenges of the Original Space

As the loss is linear in terms of the depths of the keys in BST, it may sound natural to work with the space of depth sequences of the keys. Concretely, let $\gamma \in \mathbb{N}^{2n+1}$ be the vector of the depths of K_i ’s and D_j ’s of a BST. Now consider the convex hull of all γ ’s in this space. To our knowledge, there is no well-known characterization of this polytope and we believe, if there is one, it will probably have exponentially many facets. This makes it impossible to directly apply CH-like algorithms to this problem in the original space. Furthermore, the algorithm of Suehiro et. al. [20] does not apply to this problem as the polytope is not a subset of a simplex, and consequently, cannot be characterized as submodular base polytope. Nevertheless, one can apply Follow the Perturbed Leader (FPT) [13] to this problem. Note that the expected search cost is bounded by $B = n$ in each trial. Since $\|\ell^{(t)}\|_1 = 1$ and $\|\gamma - \gamma'\| < cn^2$ for some c , then we can obtain the regret bound below:

$$\mathbb{E}[L_{\text{FPL}}] - L^* \leq 2\sqrt{c}n^{\frac{3}{2}}\sqrt{T}$$

5.2 The Dynamic Programming Game

We now go over the modules of the underlying dynamic programming game. We use the prefix “DP-” in our abbreviations to emphasize the use of the algorithm in the context of dynamic programming game rather than the original space. The optimal BST problem can be solved via dynamic

⁵It is fairly straight-forward to see if $\ell_e \in [0, b]$ for some b , the regret bounds for CH and EH will scale up accordingly.

⁶We assume the root has depth 1.

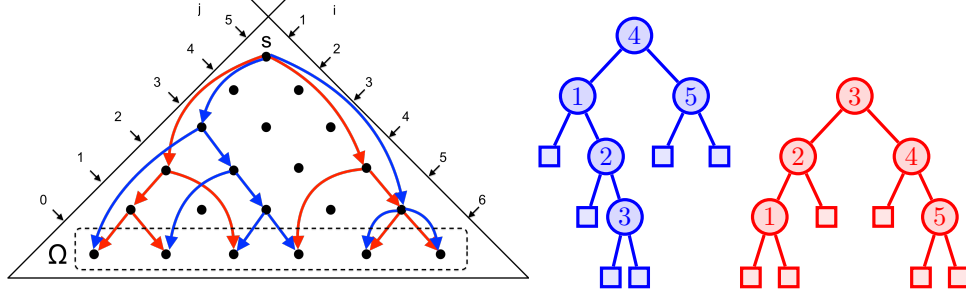


Figure 2: (left) Two different 2-multipaths in the DAG, in red and blue, and (right) their associated BSTs of $n = 5$ keys and 6 “dummy” keys. Note that each node, and consequently edge, is visited at most once in these 2-multipaths.

Problem	FPL	DP-EH	DP-CH
Optimal Binary Search Tree	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{T})$
Matrix-Chain Multiplications	—	$\mathcal{O}(n^{\frac{3}{2}} M^3 \sqrt{T})$	$\mathcal{O}(n (\log n)^{\frac{1}{2}} M^3 \sqrt{T})$
Knapsack	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n (\log nC)^{\frac{1}{2}} \sqrt{T})$
Rod Cutting	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{T})$
Weighted Interval Scheduling	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n^{\frac{3}{2}} \sqrt{T})$	$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{T})$

Table 1: Performance of various algorithms over different problems. C is the capacity in the Knapsack problem, and M is the upper-bound on the dimension in matrix-chain multiplication problem.

programming [6] with the recurrence relation below:

$$\text{OPT}(i, j) = \begin{cases} q_{i-1} & j = i - 1 \\ \min_{i \leq r \leq j} \{ \text{OPT}(i, r - 1) + \text{OPT}(r + 1, j) + \sum_{a=i}^r p_a^{(t)} + \sum_{a=r+1}^j q_a^{(t)} \} & i \leq j \end{cases} \quad (4)$$

in which $\text{OPT}(i, j)$ indicates the expected search cost of optimal BST given the keys K_i, \dots, K_j and dummy keys D_{i-1}, \dots, D_j . Note that given the recurrence relation (4), we have $k = 2$ and $s = (1, n)$. Moreover:

$$V = \{(i, j) | 1 \leq i \leq n + 1, \quad i - 1 \leq j \leq n\}, \quad \Omega = \{(i, j) | j = i - 1\}$$

$$\mathcal{F}_{(i, j)} = \{((i, j), (i, r - 1)), ((i, j), (r + 1, j))\} \mid i \leq r \leq j\}$$

Note that the loss associated with each 2-tuple is upper-bounded by $\sum_{i=1}^n p_i^{(t)} + \sum_{i=0}^n q_i^{(t)} = 1$. Figure 2 illustrates the underlying 2-regular DAG and 2-multipaths associated with BSTs.

Regret Bound It is well-known that the number of binary trees with n nodes is the n th Catalan number [6]. Therefore $\mathcal{N} = \frac{(2n)!}{n!(n+1)!} \in (2^n, 4^n)$. Also note that the expected search cost is bounded by $B = n$ in each trial. Thus using Theorem 3 we obtain:

$$\mathbb{E}[L_{\text{DP-EH}}] - L^* \leq \sqrt{2 \ln(4)} n^{\frac{3}{2}} \sqrt{T} + n^2 \ln(4)$$

Additionally, notice that each 2-multipath associated with a BST consists of exactly $D = 2n$ edges. Also we have $|V| = \frac{(n+1)(n+2)}{2}$. Therefore using Theorem 4 we obtain:

$$\mathbb{E}[L_{\text{DP-CH}}] - L^* \leq 4\sqrt{2} n (\log n)^{\frac{1}{2}} \sqrt{T} + 16 n \log n$$

6 Conclusions and Future Work

We developed a general framework for online learning of combinatorial objects whose offline optimization problems can be efficiently solved via a large class of dynamic programming algorithms.

Several examples of such objects are discussed in the main body and Appendix of this paper. Table 1 shows the performance of EH and CH in our framework (denoted by the prefix “DP-”) on different problems. See the Appendix A for more details.

In Expanded Hedge, projections are exact (i.e. renormalizing a weight vector). In contrast, iterative Bregman projections are often used for algorithms like Component Hedge [12, 15]. These methods are known to converge to the exact projection theoretically [4, 5] and are reported to be empirically very efficient [15]. However, the iterative nature of the projection step necessitates an analysis such as the one in Appendix E to bound the additional loss incurred due to stopping short of full convergence.

There are still several instances that are not included in our class of dynamic programming algorithms. The notable examples of such dynamic programming algorithms are Viterbi algorithm for finding the most probable path in a graph, and variants of Cocke-Younger-Kasami (CYK) algorithm for parsing for stochastic context-free grammars. Extending our method to these dynamic programming algorithms can be an active area of future work.

Acknowledgments

Holakou Rahmanian and Manfred K. Warmuth have been supported by NSF grant IIS-1619271.

References

- [1] Nir Ailon. Improved bounds for online learning over the permutahedron and other ranking polytopes. In *AISTATS*, pages 29–37, 2014.
- [2] Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2013.
- [3] Baruch Awerbuch and Robert Kleinberg. Online linear optimization and adaptive routing. *Journal of Computer and System Sciences*, 74(1):97–114, 2008.
- [4] 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.
- [5] 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.
- [6] Thomas H. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press Cambridge, 2009.
- [7] Corinna Cortes, Vitaly Kuznetsov, Mehryar Mohri, and Manfred Warmuth. On-line learning algorithms for path experts with non-additive losses. In *Conference on Learning Theory*, pages 424–447, 2015.
- [8] Varsha Dani, Sham M Kakade, and Thomas P Hayes. The price of bandit information for online optimization. In *Advances in Neural Information Processing Systems*, pages 345–352, 2008.
- [9] 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.
- [10] Swati Gupta, Michel Goemans, and Patrick Jaillet. Solving combinatorial games using products, projections and lexicographically optimal bases. *arXiv preprint arXiv:1603.00522*, 2016.
- [11] András György, Tamás Linder, Gábor Lugosi, and György Ottucsák. The on-line shortest path problem under partial monitoring. *Journal of Machine Learning Research*, 8(Oct):2369–2403, 2007.
- [12] David P Helmbold and Manfred K Warmuth. Learning permutations with exponential weights. *The Journal of Machine Learning Research*, 10:1705–1736, 2009.
- [13] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.
- [14] Jon Kleinberg and Eva Tardos. *Algorithm design*. Addison Wesley, 2006.
- [15] Wouter M Koolen, Manfred K Warmuth, and Jyrki Kivinen. Hedging structured concepts. 2010.
- [16] Dima Kuzmin and Manfred K Warmuth. Optimum follow the leader algorithm. In *Learning Theory*, pages 684–686. Springer, 2005.
- [17] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.
- [18] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.
- [19] Arun Rajkumar and Shivani Agarwal. Online decision-making in general combinatorial spaces. In *Advances in Neural Information Processing Systems*, pages 3482–3490, 2014.
- [20] 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.
- [21] Eiji Takimoto and Manfred K Warmuth. Path kernels and multiplicative updates. *The Journal of Machine Learning Research*, 4:773–818, 2003.

- [22] 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.
- [23] 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.

A More Instantiations

We apply our method on a few more instances of combinatorial objects recognized by dynamic programming .

A.1 Matrix Chain Multiplication

Assume a sequence A_1, A_2, \dots, A_n of n matrices to be multiplied and our goal is to compute the product $A_1 \times A_2 \times \dots \times A_n$. Using the standard algorithm for multiplying pairs of matrices as a subroutine, we can find this product by a *full parenthesizing* which basically specifies the order which the matrices are multiplied together. Concretely, we say a product of matrices is fully parenthesized if it is either a single matrix or the multiplication of two fully parenthesized matrix products, surrounded by parentheses. For instance, there are five distinct ways of fully parenthesizing the product $A_1 A_2 A_3 A_4$:

$$\begin{aligned} & (A_1(A_2(A_3A_4))) \\ & (A_1((A_2A_3)A_4)) \\ & ((A_1A_2)(A_3A_4)) \\ & (((A_1A_2)A_3)A_4) \\ & ((A_1(A_2A_3))A_4) \end{aligned}$$

Now consider the following online version of *matrix-chain multiplication* problem [6]. We are given a chain A_1, A_2, \dots, A_n of n matrices where for each $i \in [n]$. In each trial t , the algorithm predicts with a *full parenthesizing* of the product $A_1 \times A_2 \times \dots \times A_n$. Then the adversary reveals the dimensions of each A_i at trial t denoted by $m_{i-1}^{(t)} \times m_i^{(t)}$. The loss is defined as the number of scalar multiplications in the matrix-chain product.

The Dynamic Programming Game We now go over the modules of the underlying dynamic programming game. The matrix-chain multiplication problem can be solved via dynamic programming [6] with the recurrence relation below:

$$\text{OPT}(i, j) = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{ \text{OPT}(i, k) + \text{OPT}(k+1, j) + m_{i-1}^{(t)} m_k^{(t)} m_j^{(t)} \} & \text{else} \end{cases} \quad (5)$$

in which $\text{OPT}(i, j)$ indicates minimum loss of the matrix product $A_i \dots A_j$. Note that given the recurrence relation (8), we have $k = 2$ and $s = (1, n)$. The DAG \mathcal{G} has the following properties:

$$\begin{aligned} V &= \{(i, j) | 1 \leq i \leq j \leq n\}, & \Omega &= \{(i, i) | 1 \leq i \leq n\} \\ \mathcal{F}_{(i, j)} &= \{ \{((i, j), (i, k)), ((i, j), (k+1, j))\} \mid i \leq k \leq j \} \end{aligned}$$

Assuming that $m_i^{(t)} < M$ for all $i \in [n]$ and all $t \in [T]$, note that the loss associated with each 2-tuple is upper-bounded by M^3 . Figure 3 illustrates the underlying 2-regular DAG and 2-multipaths associated with full parenthesizing.

Regret Bound It is well-known that the number of full parenthesizing of a sequence of n matrices is the n th Catalan number [6]. Therefore $\mathcal{N} = \frac{(2n)!}{n!(n+1)!} \in (2^n, 4^n)$. Also note that the number of scalar multiplications in each full parenthesizing is bounded by $B = (n-1)M^3$ in each trial. Thus using Theorem 3 we obtain:

$$\mathbb{E}[L_{\text{DP-EH}}] - L^* = \mathcal{O}(n^{\frac{3}{2}} M^3 \sqrt{T}) \quad (6)$$

Additionally, notice that each 2-multipath associated with a full parenthesizing consists of exactly $D = 2(n-1)$ edges. Also we have $|V| = \frac{n(n+1)}{2}$. Therefore using Theorem 4 we obtain:

$$\mathbb{E}[L_{\text{DP-CH}}] - L^* = \mathcal{O}(n (\log n)^{\frac{1}{2}} M^3 \sqrt{T}) \quad (7)$$

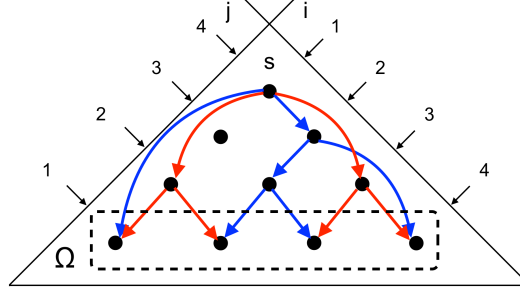


Figure 3: Given a chain of $n = 4$ matrices, the 2-multipaths associated with full parenthesizings $((A_1 A_2)(A_3 A_4))$ and $(A_1((A_2 A_3)A_4))$ are depicted in red and blue, respectively.

A.2 Knapsack

Consider the online version of *knapsack* problem [14]. Concretely, we are given a set of n items along with their *weight/heaviness* as a vector $\mathbf{h} \in \mathbb{N}^n$ as well as the *capacity* of the knapsack $C \in \mathbb{N}$. In each trial t , the algorithm predicts with a *packing* $\gamma^{(t-1)} \in \{0, 1\}^n | \mathbf{h} \cdot \gamma \leq C\}$ i.e. a subset of items whose total weight is at most the capacity of the knapsack. To make the problem interesting, we assume C and \mathbf{h} are in such way that the number of feasible packings is $\mathcal{O}(2^n)$. Then the adversary reveals a profit vector $\mathbf{p}^{(t)} \in [0, 1]^n$ in which the i th component $p_i^{(t)}$ is the profit of the i th item at trial t . The gain is defined as the sum of the profits of the items in the knapsack which is $\gamma^{(t-1)} \cdot \mathbf{p}^{(t)}$.

Challenges of the Original Space As the gain is linear in the space of γ 's, it may sound natural to work with this space. Now consider the convex hull of $\{\gamma \in \{0, 1\}^n | \mathbf{h} \cdot \gamma \leq C\}$. To our knowledge, there is no well-known characterization of this polytope and we believe, if there is one, depending on C and \mathbf{h} , it may have exponentially many facets. This makes it impossible to directly apply CH-like algorithms to this problem in the original space. Furthermore, the algorithm of Suehiro et. al. [20] does not apply to this problem as the polytope is not a subset of a simplex, and consequently, cannot be characterized as submodular base polytope.

Nevertheless, one can apply Follow the Perturbed Leader (FPT) [13] to this problem. Note that the profit is bounded by $B = n$ in each trial. Since $\|\ell^{(t)}\|_1 \leq n$ and $\|\gamma - \gamma'\| \leq n$, then we can obtain the regret bound below:

$$\mathbb{E}[L_{\text{FPL}}] - L^* \leq 2n^{\frac{3}{2}}\sqrt{T}$$

The Dynamic Programming Game We now go over the modules of the underlying dynamic programming game. The knapsack problem can be solved via dynamic programming [14] with the recurrence relation below:

$$\text{OPT}(i, c) = \begin{cases} 0 & i = 0 \\ \text{OPT}(i-1, c) & c < h_i \\ \max\{\text{OPT}(i-1, c), p_i^{(t)} + \text{OPT}(i-1, c - h_i)\} & \text{else} \end{cases} \quad (8)$$

in which $\text{OPT}(i, c)$ indicates maximum profit given items $1, \dots, i$ and capacity c . Note that given the recurrence relation (8), we have $k = 1$ and $s = (n, C)$. The DAG \mathcal{G} has the following properties:

$$V = \{(i, c) | 0 \leq i \leq n, \quad 0 \leq c \leq C\}, \quad \Omega = \{(0, c) | 0 \leq c \leq C\}$$

$$\mathcal{F}_{(i, c)} = \{\{(i, c), (i-1, c)\}\}, \{\{(i, c), (i-1, c - h_i)\}\}$$

This is basically the online longest-path problem with several sink nodes. Note that the gain associated with each edge is upper-bounded by 1. Figure 4 illustrates the underlying DAG and paths associated with packings. We turn the problem into shortest-path problem by defining a loss for each edge $e \in E$ as $\ell_e = 1 - g_e$ in which g_e is the gain of e obtained from (8). Call this new DAG $\tilde{\mathcal{G}}$. Observe that since all the paths contain n edges, the loss of each path π in $\tilde{\mathcal{G}}$ (denoted by $L_{\tilde{\mathcal{G}}}(\pi)$) is directly connected to the gain of π in \mathcal{G} (denoted by $G_{\mathcal{G}}(\pi)$) by $L_{\tilde{\mathcal{G}}}(\pi) = n - G_{\mathcal{G}}(\pi)$.

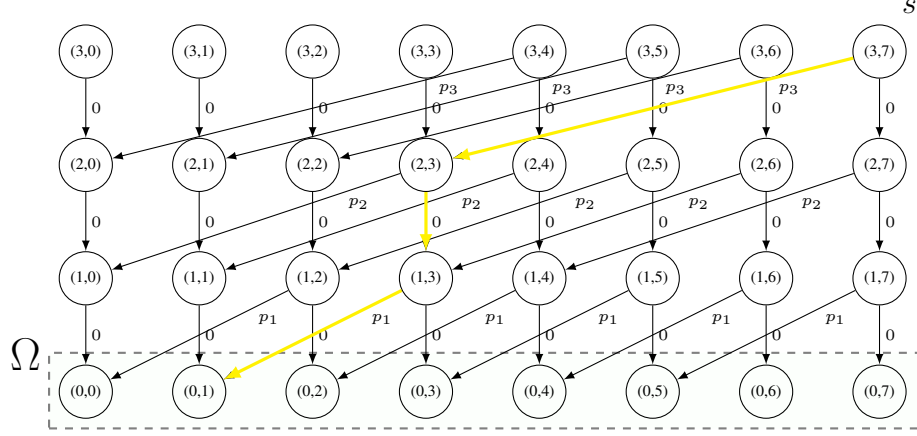


Figure 4: Example with $C = 7$ and $\mathbf{h} = [2, 3, 4]$. Packing $(1, 0, 1)$ is highlighted.

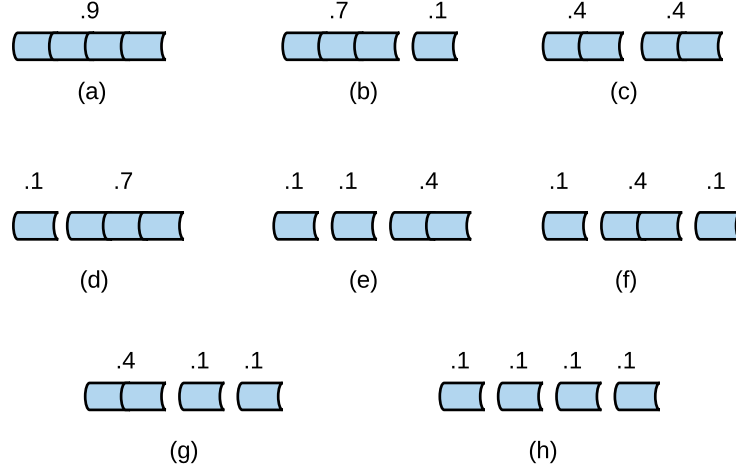


Figure 5: All cuttings of a rod of length $n = 4$ and their profits given profit vector $\mathbf{p} = (.1, .4, .7, .9)$.

Regret Bounds According to our initial assumption $\mathcal{N} = \mathcal{O}(2^n)$. Also note that loss of each path in each trial is bounded by $B = n$. Thus using Theorem 3 we obtain:

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-EH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-EH}}]) = \mathbb{E}[L_{\text{DP-EH}}] - L^* \\ &= \mathcal{O}(n^{\frac{3}{2}} \sqrt{T}) \end{aligned}$$

Notice that the number of vertices is $|V| = nC$ and each path consists of $D = n$ edges. Therefore using Theorem 4 we obtain:

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-CH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-CH}}]) = \mathbb{E}[L_{\text{DP-CH}}] - L^* \\ &= \mathcal{O}(n (\log nC)^{\frac{1}{2}} \sqrt{T}) \end{aligned}$$

A.3 Rod Cutting

Consider the online version of *rod cutting* problem [6]. A rod of length n is given. In each trial t , the algorithm predicts with a *cutting* $\gamma^{(t-1)}$, that is, it cuts up the rod into pieces. Mathematically speaking, $\gamma^{(t-1)} \in \{\gamma \in \mathbb{N}^n \mid \gamma \cdot [1, 2, \dots, n] = n\}$. Then the adversary reveals a price vector $\mathbf{p}^{(t)} \in [0, 1]^n$ in which the i th component $p_i^{(t)}$ is the price of the piece of length i at trial t . The gain is defined as the total sales in the trial which is $\gamma_{t-1} \cdot \mathbf{p}_t$. See Figure 5 as an example.

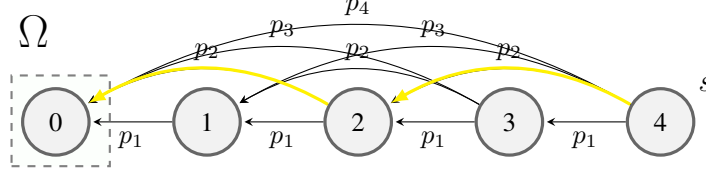


Figure 6: Example with $n = 4$. Cutting $(0, 2, 0, 0)$ is highlighted.

Challenges of the Original Space As the gain is linear in the space of γ 's, it may sound natural to work with this space. Now consider the convex hull of $\{\gamma \in \mathbb{N}^n \mid \gamma \cdot [1, 2, \dots, n] = n\}$. To our knowledge, there is no well-known characterization of this polytope and we believe, if there is one, it will probably have exponentially many facets. This makes it impossible to directly apply CH-like algorithms to this problem in the original space. Furthermore, the algorithm of Suehiro et. al. [20] does not apply to this problem as the polytope is not a subset of a simplex, and consequently, cannot be characterized as submodular base polytope.

Nevertheless, one can apply Follow the Perturbed Leader (FPT) [13] to this problem. Note that the profit is bounded by $B = n$ in each trial. Since $\|\ell^{(t)}\|_1 \leq n$ and $\|\gamma - \gamma'\| \leq n$, then we can obtain the regret bound below:

$$\mathbb{E}[L_{\text{FPL}}] - L^* \leq 2n^{\frac{3}{2}}\sqrt{T}$$

The Dynamic Programming Game We now go over the modules of the underlying dynamic programming game. The rod cutting problem can be solved via dynamic programming [6] with the recurrence relation below:

$$\text{OPT}(i) = \begin{cases} 0 & i = 0 \\ \max_{0 \leq j \leq i} \{\text{OPT}(j) + p_{j-i}^{(t)}\} & i > 0 \end{cases} \quad (9)$$

in which $\text{OPT}(i)$ indicates maximum profit given a rod of length i . Note that given the recurrence relation (9), we have $k = 1$ and $s = n$. The DAG \mathcal{G} has the following properties:

$$\begin{aligned} V &= \{i \mid 0 \leq i \leq n\}, & \Omega &= \{0\} \\ \mathcal{F}_i &= \{(i, j) \mid 0 \leq j \leq i\} \end{aligned}$$

This is basically the online longest-path problem with one sink node. Note that the gain associated with each edge is upper-bounded by 1. Figure 6 illustrates the underlying DAG and paths associated with cuttings. Similar to the knapsack problem, we turn this problem into shortest-path problem as well. In order to do so, we first need to modify the graph in a way that all paths have equal length of n (the length of the longest path) and the gain of each path remains fixed. Using the technique introduced in György et. al. [11], we can add $\mathcal{O}(n^2)$ vertices and edges (with gain zero) to make all paths equi-length. Then, we define a loss for each edge e as $\ell_e = 1 - g_e$ in which g_e is the gain of e . Call this new DAG $\bar{\mathcal{G}}$. Similar to the knapsack problem, we have $L_{\bar{\mathcal{G}}}(\pi) = n - G_{\bar{\mathcal{G}}}(\pi)$ for all paths π .

Regret Bounds Note that in both \mathcal{G} and $\bar{\mathcal{G}}$, there are $\mathcal{N} = 2^{n-1}$ paths. Also note that loss of each path in each trial is bounded by $B = n$. Thus using Theorem 3 we obtain ⁷

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-EH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-EH}}]) = \mathbb{E}[L_{\text{DP-EH}}] - L^* \\ &= \mathcal{O}(n^{\frac{3}{2}}\sqrt{T}) \end{aligned}$$

Notice that the number of vertices in $\bar{\mathcal{G}}$ is $\mathcal{O}(n^2)$ and each path consists of $D = n$ edges. Therefore using Theorem 4 we obtain:

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-CH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-CH}}]) = \mathbb{E}[L_{\text{DP-CH}}] - L^* \\ &= \mathcal{O}(n(\log n)^{\frac{1}{2}}\sqrt{T}) \end{aligned}$$

⁷We are over-counting the number of cuttings. The number of possible cutting is called **partition function** which is approximately $e^{\pi\sqrt{2n/3}}/4n\sqrt{3}$ [6]. Thus if we naively apply EH algorithm in an inefficient way, we will get better regret bound by a factor of $\sqrt[4]{n}$.

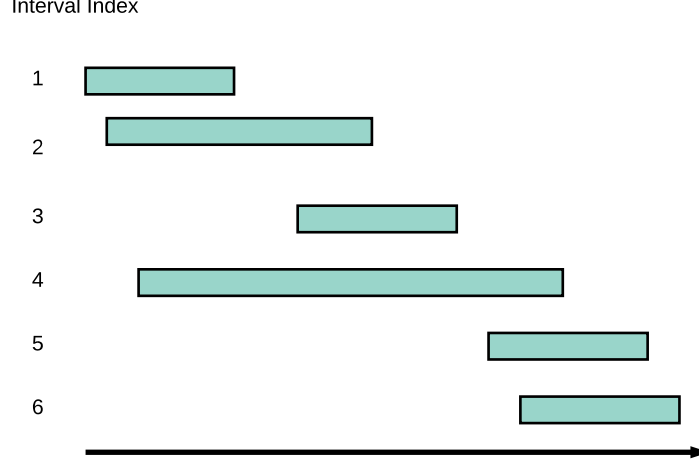


Figure 7: Example with $n = 6$

A.4 Weighted Interval Scheduling

Consider the online version of *weighted interval scheduling* problem [14]. We are given a set of n intervals I_1, \dots, I_n on the real line. In each trial t , the algorithm predicts with a *scheduling* $\gamma^{(t-1)} \in \{0, 1\}^n$, that is a subset of non-overlapping intervals. Then the adversary reveals a profit vector $\mathbf{p}^{(t)} \in [0, 1]^n$ in which the i th component $p_i^{(t)}$ is the profit of including I_i in the scheduling at trial t . The gain is defined as the total profit over chosen intervals in the scheduling in the trial which is $\gamma_{t-1} \cdot \mathbf{p}_t$. See Figure 7 as an example.

Challenges of the Original Space As the gain is linear in the space of γ 's, it may sound natural to work with this space. Now consider the convex hull of $\{\gamma \in \{0, 1\}^n \mid \text{intervals } i \text{ with } \gamma_i = 1 \text{ don't overlap}\}$. To our knowledge, there is no well-known characterization of this polytope and we believe, if there is one, depending on the given intervals, it may have exponentially many facets. This makes it impossible to directly apply CH-like algorithms to this problem in the original space. Furthermore, the algorithm of Suehiro et. al. [20] does not apply to this problem as the polytope is not a subset of a simplex, and consequently, cannot be characterized as submodular base polytope.

Nevertheless, one can apply Follow the Perturbed Leader (FPT) [13] to this problem. Note that the profit is bounded by $B = n$ in each trial. Since $\|\ell^{(t)}\|_1 \leq n$ and $\|\gamma - \gamma'\| < n$, then we can obtain the regret bound below:

$$\mathbb{E}[L_{\text{FPL}}] - L^* \leq 2n^{\frac{3}{2}}\sqrt{T}$$

The Dynamic Programming Game We now go over the modules of the underlying dynamic programming game. The weighted interval scheduling problem can be solved via dynamic programming [14]. Defining $\text{OPT}(i)$ as the maximum profit given the intervals I_1, \dots, I_i , the recurrence relation below holds:

$$\text{OPT}(i) = \begin{cases} 0 & i = 0 \\ \max\{\text{OPT}(i-1), \text{OPT}(\text{pred}(i)) + p_i^{(t)}\} & i > 0 \end{cases} \quad (10)$$

in which

$$\text{pred}(i) := \begin{cases} 0 & i = 1 \\ \max_{\{j < i, I_i \cap I_j = \emptyset\}} j & i > 1 \end{cases}$$

Note that given the recurrence relation (10), we have $k = 1$ and $s = n$. The DAG \mathcal{G} has the following properties:

$$\begin{aligned} V &= \{i \mid 0 \leq i \leq n\}, & \Omega &= \{0\} \\ \mathcal{F}_i &= \{\{(i, i-1)\}, \{(i, \text{pred}(i))\}\} \end{aligned}$$

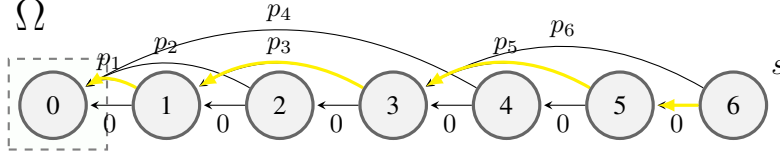


Figure 8: The underlying DAG associated with the example illustrated in Figure 7. Scheduling $(1, 0, 1, 0, 1, 0)$ is highlighted.

Similar to rod cutting, this is also the online longest-path problem with one sink node. Note that the gain associated with each edge is upper-bounded by 1. Figure 8 illustrates the underlying DAG and paths associated with scheduling. Like the rod cutting problem, we modify the graph by adding $\mathcal{O}(n^2)$ vertices and edges (with gain zero) to make all paths equi-length and change the gains into losses. Call this new DAG $\bar{\mathcal{G}}$. Again we have $L_{\bar{\mathcal{G}}}(\pi) = n - G_{\bar{\mathcal{G}}}(\pi)$ for all paths π .

Regret Bounds According to our initial assumption $\mathcal{N} = \mathcal{O}(2^n)$. Also note that loss of each path in each trial is bounded by $B = n$. Thus using Theorem 3 we obtain:

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-EH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-EH}}]) = \mathbb{E}[L_{\text{DP-EH}}] - L^* \\ &= \mathcal{O}(n^{\frac{3}{2}} \sqrt{T}) \end{aligned}$$

Notice that the number of vertices in $\bar{\mathcal{G}}$ is $\mathcal{O}(n^2)$ and each path consists of $D = n$ edges. Therefore using Theorem 4 we obtain:

$$\begin{aligned} G^* - \mathbb{E}[G_{\text{DP-CH}}] &= (nT - L^*) - (nT - \mathbb{E}[L_{\text{DP-CH}}]) = \mathbb{E}[L_{\text{DP-CH}}] - L^* \\ &= \mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{T}) \end{aligned}$$

B Generalized Weight Pushing Correctness

Lemma 5. *The weights $w_{\tilde{E}}^{(t)}$ generated by the generalized weight pushing satisfies the Φ properties and $W_{\pi}^{(t)} \propto \prod_{\tilde{E} \in \mathcal{F}} (\hat{w}_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}}$. Moreover, the weights $w_{\tilde{E}}^{(t)}$ can be computed in $\mathcal{O}(|E|)$ time.*

Proof For all $v \in V$, Z_v is defined as the normalization constant if v was the source in \mathcal{G} . Concretely, let P_v be the set of all k -multipaths sourced from v and sinking at Ω . Then:

$$Z_v = \sum_{\pi_v \in P_v} W_{\pi_v}^{(t-1)} \exp \left[-\eta \pi_v \cdot \ell^{(t)} \right]$$

For a sink node $v \in \Omega$, the normalization constant is vacuously 1 since no normalization is needed. For other $v \in V - \Omega$, we partition the summation by the starting k -tuple from v :

$$\begin{aligned} Z_v &= \sum_{\pi_v \in P_v} W_{\pi_v}^{(t-1)} \exp \left[-\eta \pi_v \cdot \ell^{(t)} \right] \\ &= \sum_{\tilde{E} \in \mathcal{F}_v} \sum_{\substack{\pi_v \in P_v \\ \text{starts with } \tilde{E}}} W_{\pi_v}^{(t-1)} \exp \left[-\eta \pi_v \cdot \ell^{(t)} \right] \end{aligned}$$

Now, we can factor out the weight and exponentiated loss associated with $\tilde{E} \in \mathcal{F}_v$. Notice, excluding \tilde{E} from the k -multipath, we are left with k number of k -multipaths from all u 's such that $(v, u) \in \tilde{E}$:

$$\begin{aligned} Z_v &= \sum_{\tilde{E} \in \mathcal{F}_v} \sum_{\substack{\pi_v \in P_v \\ \text{starts with } \tilde{E}}} W_{\pi_v}^{(t-1)} \exp \left[-\eta \pi_v \cdot \ell^{(t)} \right] \\ &= \sum_{\tilde{E} \in \mathcal{F}_v} \underbrace{\left(w_{\tilde{E}}^{(t-1)} \right)^{\pi_{\tilde{E}}}}_{\hat{w}_{\tilde{E}}^{(t)}} \exp \left[-\eta \pi_{\tilde{E}} \sum_{e \in \tilde{E}} \ell_e^{(t)} \right] \sum_{\substack{\{\pi_u\}_{u:(v,u) \in \tilde{E}} \\ \text{s.t. } \forall u \pi_u \in P_u}} \prod_{u:(v,u) \in \tilde{E}} W_{\pi_u}^{(t-1)} \exp \left[-\eta \pi_u \cdot \ell^{(t)} \right] \end{aligned}$$

Observe that since the π_u 's are independent for different u 's, we can turn the sum of products into product of sums:

$$\begin{aligned}
Z_v &= \sum_{\tilde{E} \in \mathcal{F}_v} \underbrace{\left(w_{\tilde{E}}^{(t-1)} \right)^{\pi_{\tilde{E}}} \exp \left[-\eta \pi_{\tilde{E}} \sum_{e \in \tilde{E}} \ell_e^{(t)} \right]}_{\hat{w}_{\tilde{E}}^{(t)}} \sum_{\substack{\{\pi_u\}_{u:(v,u) \in \tilde{E}} \\ \text{s.t. } \forall u \pi_u \in P_u}} \prod_{u:(v,u) \in \tilde{E}} W_{\pi_u}^{(t-1)} \exp \left[-\eta \pi_u \cdot \ell^{(t)} \right] \\
&= \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \prod_{u:(v,u) \in \tilde{E}} \underbrace{\sum_{\pi_u \in P_u} W_{\pi_u}^{(t-1)} \exp \left[-\eta \pi_u \cdot \ell^{(t)} \right]}_{Z_u} \\
&= \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \prod_{u:(v,u) \in \tilde{E}} Z_u
\end{aligned}$$

Now for each $v \in V - \Omega$, for all $\tilde{E} \in \mathcal{F}_v$, set $w_{\tilde{E}}^{(t)} := \hat{w}_{\tilde{E}}^{(t)} \frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v}$ and let $W_{\pi}^{(t)} := \prod_{\tilde{E} \in \mathcal{F}} (w_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}}$. Φ_1 becomes immediately true because of the definition of $W_{\pi}^{(t)}$. Φ_2 is also true since:

$$\begin{aligned}
\sum_{\tilde{E} \in \mathcal{F}_v} w_{\tilde{E}}^{(t)} &= \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v} \\
&= \frac{1}{Z_v} \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \prod_{u:(v,u) \in \tilde{E}} Z_u \\
&= \frac{1}{Z_v} \times Z_v = 1
\end{aligned}$$

For $W_{\pi}^{(t)}$, we can write :

$$\begin{aligned}
W_{\pi}^{(t)} &= \prod_{\tilde{E} \in \mathcal{F}} (w_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}} = \prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} (w_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}} \\
&= \prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\hat{w}_{\tilde{E}}^{(t)} \frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v} \right)^{\pi_{\tilde{E}}} \\
&= \left[\prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\hat{w}_{\tilde{E}}^{(t)} \right)^{\pi_{\tilde{E}}} \right] \left[\prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v} \right)^{\pi_{\tilde{E}}} \right]
\end{aligned}$$

Notice that $\prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v} \right)^{\pi_{\tilde{E}}}$ telescopes along the k -tuples in the k -multipath. Therefore we obtain:

$$\begin{aligned}
W_{\pi}^{(t)} &= \left[\prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\hat{w}_{\tilde{E}}^{(t)} \right)^{\pi_{\tilde{E}}} \right] \left[\prod_{v \in V - \Omega} \prod_{\tilde{E} \in \mathcal{F}_v} \left(\frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v} \right)^{\pi_{\tilde{E}}} \right] \\
&= \left[\prod_{\tilde{E} \in \mathcal{F}} \left(\hat{w}_{\tilde{E}}^{(t)} \right)^{\pi_{\tilde{E}}} \right] \left[\frac{\prod_{v \in \Omega} Z_v}{Z_s} \right] \\
&= \frac{1}{Z_s} \prod_{\tilde{E} \in \mathcal{F}} \left(\hat{w}_{\tilde{E}}^{(t)} \right)^{\pi_{\tilde{E}}}
\end{aligned}$$

which indicates that $W_{\pi}^{(t)} \propto \prod_{\tilde{E} \in \mathcal{F}} (\hat{w}_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}}$. Finally Φ_3 is true as well because:

$$\begin{aligned} \sum_{\pi} W_{\pi}^{(t)} &= \frac{1}{Z_s} \prod_{\tilde{E} \in \mathcal{F}} (\hat{w}_{\tilde{E}}^{(t)})^{\pi_{\tilde{E}}} \\ &= \frac{1}{Z_s} \times Z_s = 1 \end{aligned}$$

Regarding the time complexity, we first focus on the recurrence relation $Z_v = \sum_{\tilde{E} \in \mathcal{F}_v} \hat{w}_{\tilde{E}}^{(t)} \prod_{u:(v,u) \in \tilde{E}} Z_u$. Note that for each $v \in V$, Z_v can be computed in $\mathcal{O}(|E_v|)$ in which E_v is the set of out-going edges from v . Thus the computation of all Z_v 's takes $\mathcal{O}(|E|)$ time. Now observe that $w_{\tilde{E}}^{(t)}$ for each $\tilde{E} \in \mathcal{F}$ can be found in $\mathcal{O}(k)$ time using $w_{\tilde{E}}^{(t)} = \hat{w}_{\tilde{E}}^{(t)} \frac{\prod_{u:(v,u) \in \tilde{E}} Z_u}{Z_v}$. Hence the computation of all $w_{\tilde{E}}^{(t)}$'s takes $\mathcal{O}(|E|)$ time since $|\mathcal{F}| \times k = |E|$. Therefore the generalized weight pushing algorithm runs in $\mathcal{O}(|E|)$.

□

C Relative Entropy Projection to k -Flow Polytope

Formally, the projection \mathbf{w} of a given point $\hat{\mathbf{w}}$ to constraint C is the solution to the following:

$$\arg \min_{\mathbf{w} \in C} \sum_{e \in E} w \log \left(\frac{w_e}{\hat{w}_e} \right) + \hat{w}_e - w_e$$

C can be three different constraints. We use the method of Lagrange multipliers in all three cases. Observe that if $k = 1$, then the third constraint is non-existent and the updates in Koolen et. al. [15] are recovered.

C.1 Root Outflow

The outflow from the root r must be k . Assume w_{r_1}, \dots, w_{r_s} are the weights associated with the outgoing edges from the root. Then:

$$\begin{aligned} L(\mathbf{w}, \lambda) &:= \sum_{e \in E} w_e \log \left(\frac{w_e}{\hat{w}_e} \right) + \hat{w}_e - w_e - \lambda \left(\sum_{j=1}^s w_{r_j} - k \right) \\ \frac{\partial L}{\partial w_e} &= \log \frac{w_e}{\hat{w}_e} = 0 \longrightarrow w_e = \hat{w}_e \quad \forall e \in E - \{r_1, \dots, r_s\} \\ \frac{\partial L}{\partial w_{r_j}} &= \log \frac{w_{r_j}}{\hat{w}_{r_j}} - \lambda = 0 \longrightarrow w_{r_j} = \hat{w}_{r_j} \exp(\lambda) \end{aligned} \tag{11}$$

$$\frac{\partial L}{\partial \lambda} = \sum_{j=1}^s w_{r_j} - k = 0 \tag{12}$$

Combining equations (11) and (12) results in normalizing w_{r_1}, \dots, w_{r_s} , that is:

$$\forall j \in [s] \quad w_{r_j} = \frac{k \hat{w}_{r_j}}{\sum_{j'=1}^s \hat{w}_{r_{j'}}}$$

C.2 k -Flow Preservation at Node v

Given any node v other than the root, the out-flow from the node v must be k times of the in-flow of that node. Assume $w_{v_1}^{(\text{in})}, \dots, w_{v_a}^{(\text{in})}$ and $w_{v_1}^{(\text{out})}, \dots, w_{v_b}^{(\text{out})}$ are the weights associated with the

incoming and outgoing edges from/to the node v , respectively. Then:

$$L(\mathbf{w}, \lambda) := \sum_{e \in E} w \log \left(\frac{w_e}{\widehat{w}_e} \right) + \widehat{w}_e - w_e - \lambda \left(\sum_{b'=1}^b w_{v_{b'}}^{(\text{out})} - k \sum_{a'=1}^a w_{v_{a'}}^{(\text{in})} \right)$$

$$\frac{\partial L}{\partial w_e} = \log \frac{w_e}{\widehat{w}_e} = 0 \longrightarrow w_e = \widehat{w}_e \quad \forall e \text{ non-adjacent to } v$$

$$\frac{\partial L}{\partial w_{v_{b'}}^{(\text{out})}} = \log \frac{w_{v_{b'}}^{(\text{out})}}{\widehat{w}_{v_{b'}}^{(\text{out})}} - \lambda = 0 \longrightarrow w_{v_{b'}}^{(\text{out})} = \widehat{w}_{v_{b'}}^{(\text{out})} \exp(\lambda) \quad \forall b' \in [b] \quad (13)$$

$$\frac{\partial L}{\partial w_{v_{a'}}^{(\text{in})}} = \log \frac{w_{v_{a'}}^{(\text{in})}}{\widehat{w}_{v_{a'}}^{(\text{in})}} + k\lambda = 0 \longrightarrow w_{v_{a'}}^{(\text{in})} = \widehat{w}_{v_{a'}}^{(\text{in})} \exp(-k\lambda) \quad \forall a' \in [a] \quad (14)$$

$$\frac{\partial L}{\partial \lambda} = \sum_{b'=1}^b w_{v_{b'}}^{(\text{out})} - k \sum_{a'=1}^a w_{v_{a'}}^{(\text{in})} = 0 \quad (15)$$

Letting $\beta = \exp(k)$, for all $a' \in [a]$ and all $b' \in [b]$, we can obtain the following by combining equations (13), (14) and (15):

$$\beta \left(\sum_{b'=1}^b \widehat{w}_{v_{b'}}^{(\text{out})} \right) = \frac{k}{\beta^k} \left(\sum_{a'=1}^a \widehat{w}_{v_{a'}}^{(\text{in})} \right) \longrightarrow \beta = \sqrt[k+1]{k \frac{\sum_{a'=1}^a \widehat{w}_{v_{a'}}^{(\text{in})}}{\sum_{b'=1}^b \widehat{w}_{v_{b'}}^{(\text{out})}}}$$

$$w_{v_{b'}}^{(\text{out})} = \widehat{w}_{v_{b'}}^{(\text{out})} \left(k \frac{\sum_{a''=1}^a \widehat{w}_{v_{a''}}^{(\text{in})}}{\sum_{b''=1}^b \widehat{w}_{v_{b''}}^{(\text{out})}} \right)^{\frac{1}{k+1}}, \quad w_{v_{a'}}^{(\text{in})} = \widehat{w}_{v_{a'}}^{(\text{in})} \left(\frac{1}{k} \frac{\sum_{b''=1}^b \widehat{w}_{v_{b''}}^{(\text{out})}}{\sum_{a''=1}^a \widehat{w}_{v_{a''}}^{(\text{in})}} \right)^{\frac{k}{k+1}}$$

This indicates that to enforce the k -flow property at each node, the weights must be multiplicatively scaled up/down so that the out- and in-flow will be proportionate to the k -to-1 weighted geometric average of the out- and in-flow, respectively. Concretely:

$$\text{out-flow} = k \times \text{in-flow} = \sqrt[k+1]{k (\widehat{\text{out-flow}})^k (\widehat{\text{in-flow}})}$$

C.3 k -Tuple Equality

Let \mathcal{F} be the set of all k -tuples of edges the components of which must have equal weights. For a given tuple $J \in \mathcal{F}$, let $w_0^{(J)}, \dots, w_{k-1}^{(J)}$ be the weights. Assuming $j^- = j - 1 \pmod{k}$ and $j^+ = j + 1 \pmod{k}$, then:

$$L(\mathbf{w}, \lambda) := \sum_{e \in E} w_e \log \left(\frac{w_e}{\widehat{w}_e} \right) + \widehat{w}_e - w_e - \sum_{J \in \mathcal{F}} \sum_{j=0}^{k-1} \lambda_j^{(J)} (w_j^{(J)} - w_{j^-}^{(J)})$$

$$\frac{\partial L}{\partial w_j^{(J)}} = \log \frac{w_j^{(J)}}{\widehat{w}_j^{(J)}} - \lambda_j^{(J)} + \lambda_{j^+}^{(J)} = 0 \longrightarrow w_j^{(J)} = \widehat{w}_j^{(J)} \frac{e^{\lambda_j^{(J)}}}{e^{\lambda_{j^+}^{(J)}}} \quad \forall j \in \{0, 1, \dots, k-1\} \quad (16)$$

$$\frac{\partial L}{\partial \lambda_j^{(J)}} = w_j^{(J)} - w_{j^-}^{(J)} = 0 \longrightarrow w_j^{(J)} = w_{j^-}^{(J)} \quad \forall j \in \{0, 1, \dots, k-1\} \quad (17)$$

Combining equations (16) and (17), for all $J \in \mathcal{F}$ and for all $j \in J$, we can obtain:

$$\left(w_j^{(J)} \right)^k = \prod_{j'=0}^{k-1} w_{j'}^{(J)} = \prod_{j'=0}^{k-1} \widehat{w}_{j'}^{(J)} \longrightarrow w_j^{(J)} = \sqrt[k]{\prod_{j'=0}^{k-1} \widehat{w}_{j'}^{(J)}}$$

which basically indicates that each weight must be assigned to the geometric average of the weights in its tuple.

D CH Regret Bound on k -Multipaths

Proof According to Koolen, Warmuth and Kivinen [15], the regret bound of CH is:

$$\mathbb{E}[L_{\text{CH}}] - L^* \leq \sqrt{2 L^* \Delta(\boldsymbol{\pi} || \mathbf{w}^{(0)})} + \Delta(\boldsymbol{\pi} || \mathbf{w}^{(0)}) \quad (18)$$

in which $\boldsymbol{\pi} \in \mathbb{N}^{|E|}$ and L^* are the best k -multipath and its loss, respectively. Let $\tilde{\mathbf{w}}^{(0)} = \frac{1}{|V|^2} \mathbf{1}$ in which $\mathbf{1} \in \mathbb{R}^{|E|}$ is a vector of all ones. Now let the initial point $\mathbf{w}^{(0)}$ be the relative entropy projection of $\tilde{\mathbf{w}}^{(0)}$ onto the k -flow polytope⁸

$$\mathbf{w}^{(0)} = \arg \min_{\mathbf{w} \in P} \Delta(\mathbf{w} || \tilde{\mathbf{w}}^{(0)})$$

Assuming $\|\boldsymbol{\pi}\|_1 \leq D \leq |E|$, we obtain:

$$\begin{aligned} \Delta(\boldsymbol{\pi} || \mathbf{w}^{(0)}) &\leq \Delta(\boldsymbol{\pi} || \tilde{\mathbf{w}}^{(0)}) && \text{Pythagorean Theorem} \\ &= \sum_{e \in E} \pi_e \log \frac{\pi_e}{\tilde{w}_i^{(0)}} + \tilde{w}_i^{(0)} - \pi_e \\ &= \sum_{e \in E} \pi_e \log \frac{1}{\tilde{w}_i^{(0)}} + \pi_e \log \pi_e + \tilde{w}_i^{(0)} - \pi_e \\ &\leq \sum_{e \in E} \pi_e (2 \log |V|) + \sum_{e \in E} \pi_e \log \pi_e + \sum_{e \in E} \frac{1}{|V|^2} - \sum_{e \in E} \pi_e \\ &\leq D(2 \log |V|) + D \log D + |E| \frac{1}{|V|^2} - \sum_{e \in E} \pi_e \\ &\leq 4 D \log |V| \end{aligned} \quad (19)$$

Thus, using inequality (18), the regret bound will be:

$$\mathbb{E}[L_{\text{CH}}] - L^* \leq \sqrt{8 L^* D \log |V|} + 4 D \log |V|$$

Note that if $\|\boldsymbol{\pi}\|_\infty = 1$, then $\sum_{e \in E} \pi_e \log \pi_e = 0$, and consequently, the expression (19) can be bound as below:

$$\begin{aligned} \Delta(\boldsymbol{\pi} || \mathbf{w}^{(0)}) &\leq \sum_{e \in E} \pi_e (2 \log |V|) + \sum_{e \in E} \pi_e \log \pi_e + \sum_{e \in E} \frac{1}{|V|^2} - \sum_{e \in E} \pi_e \\ &\leq D(2 \log |V|) + |E| \frac{1}{|V|^2} - \sum_{e \in E} \pi_e \\ &\leq 2 D \log |V| \end{aligned}$$

Again, using inequality (18), the regret bound will be:

$$\mathbb{E}[L_{\text{CH}}] - L^* \leq \sqrt{4 L^* D \log |V|} + 2 D \log |V|$$

□

E Additional Loss with Approximate Projection

First, let us define the notation \preceq . Given two vectors \mathbf{a} and \mathbf{b} , we say $\mathbf{a} \preceq \mathbf{b}$ iff every element of \mathbf{a} is less than or equal \mathbf{b} .

Now let us discuss approximate projection and additional loss. As we are working with in-exact projection, we propose a slightly different prediction algorithm for Component Hedge. Suppose, using iterative Bregman projections, we reached at $\hat{\mathbf{w}} \in \mathbb{R}_+^{|\mathcal{F}|}$ which is ϵ -close to the exact projection $\mathbf{w} \in \mathbb{R}_+^{|\mathcal{F}|}$ in 1-norm, that is $\|\mathbf{w} - \hat{\mathbf{w}}\|_1 < \epsilon$. Then do the following steps for prediction:

⁸This computation can be done as a pre-processing step.

1. Set $\tilde{\mathbf{w}} := \hat{\mathbf{w}} + \epsilon \cdot \mathbf{1}$ where $\mathbf{1} \in \mathbb{R}_+^{|\mathcal{F}|}$ is a vector of all ones.
2. Apply decomposition procedure on $\tilde{\mathbf{w}}$ and obtain Π . Since $\tilde{\mathbf{w}}$ does not necessarily belongs to the polytope Ψ , the decomposition Π will not zero-out all the edges in $\tilde{\mathbf{w}}$:

$$\tilde{\mathbf{w}} \succeq \sum_{\pi \in \Pi} p_\pi \cdot \pi = \bar{\mathbf{w}}$$

3. Normalize and sample from decomposition Π .

According to the definition, $\tilde{\mathbf{w}} \succeq \mathbf{w}$. Thus \mathbf{w} can be subtracted out from $\tilde{\mathbf{w}}$ in the decomposition and we have $\bar{\mathbf{w}} \succeq \mathbf{w}$. Therefore:

$$\mathbf{w} \preceq \bar{\mathbf{w}} \preceq \tilde{\mathbf{w}} = \hat{\mathbf{w}} + \epsilon \cdot \mathbf{1} \quad (20)$$

Now let z be the normalization constant for Π . Hence the expected prediction will be $\bar{\mathbf{w}}/z$. Note that since $\bar{\mathbf{w}} \succeq \mathbf{w}$ and $\mathbf{w} \in \Psi$, then $z \geq 1$. Also notice that the weights of out-going k -tuple from the source s in $\hat{\mathbf{w}}$ are at most 2ϵ greater than the ones in \mathbf{w} which belongs to Φ . Thus the out-flow at s in $\tilde{\mathbf{w}}$ is at most $k + 2n\epsilon$. Therefore, since $\bar{\mathbf{w}} \preceq \tilde{\mathbf{w}}$, we have $z \leq 1 + \frac{2n}{k}\epsilon$. Now we establish a closeness property between the approximate projected vector and the expected prediction vector with approximate projection :

$$\begin{aligned} \left\| \frac{\bar{\mathbf{w}}}{z} - \hat{\mathbf{w}} \right\|_1 &\leq \left\| \bar{\mathbf{w}} - \hat{\mathbf{w}} \right\|_1 + \frac{z-1}{z} \left\| \bar{\mathbf{w}} \right\|_1 \\ &\leq \left\| \bar{\mathbf{w}} - \hat{\mathbf{w}} \right\|_1 + \frac{2n}{k} \epsilon \left\| \bar{\mathbf{w}} \right\|_1 \\ &\leq \epsilon |\mathcal{F}| + \frac{2n}{k} \epsilon |\mathcal{F}| = \epsilon |\mathcal{F}| \left(1 + \frac{2n}{k}\right) \end{aligned} \quad (20)$$

Next we establish closeness between the expected prediction vectors in exact and approximate projections:

$$\begin{aligned} \left\| \frac{\bar{\mathbf{w}}}{z} - \mathbf{w} \right\|_1 &\leq \left\| \frac{\bar{\mathbf{w}}}{z} - \hat{\mathbf{w}} \right\|_1 + \left\| \hat{\mathbf{w}} - \mathbf{w} \right\|_1 \\ &\leq \epsilon |\mathcal{F}| \left(1 + \frac{2n}{k}\right) + \epsilon = \epsilon \left(1 + |\mathcal{F}| + \frac{2n}{k} |\mathcal{F}|\right) \end{aligned}$$

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

$$\begin{aligned} \left\| \sum_{t=1}^T \frac{\bar{\mathbf{w}}^{(t)}}{z} \cdot \boldsymbol{\ell}^{(t)} \right\|_1 &\leq \left\| \sum_{t=1}^T \mathbf{w}^{(t)} \cdot \boldsymbol{\ell}^{(t)} \right\|_1 + \left\| \sum_{t=1}^T \left(\frac{\bar{\mathbf{w}}^{(t)}}{z} - \mathbf{w}^{(t)} \right) \cdot \boldsymbol{\ell}^{(t)} \right\|_1 \\ &\leq \left\| \sum_{t=1}^T \mathbf{w}^{(t)} \cdot \boldsymbol{\ell}^{(t)} \right\|_1 + \sum_{t=1}^T \left\| \frac{\bar{\mathbf{w}}^{(t)}}{z} - \mathbf{w}^{(t)} \right\|_2 \cdot \left\| \boldsymbol{\ell}^{(t)} \right\|_2 \\ &\leq \left\| \sum_{t=1}^T \mathbf{w}^{(t)} \cdot \boldsymbol{\ell}^{(t)} \right\|_1 + T \times \epsilon \left(1 + |\mathcal{F}| + \frac{2n}{k} |\mathcal{F}|\right) \times \sqrt{|\mathcal{F}|} \end{aligned}$$

Setting $\epsilon = \frac{1}{T(1+|\mathcal{F}|+\frac{2n}{k}|\mathcal{F}|)\sqrt{|\mathcal{F}|}}$ we add at most one unit to the expected cumulative loss with exact projection.