Efficient Algorithms for Online Combinatorial Optimization and Its Application to Learning over Compressed Data

藤田, 隆寛

https://doi.org/10.15017/1931928

出版情報：九州大学, 2017, 博士（情報科学）, 課程博士
バージョン：
権利関係：

Efficient Algorithms for Online Combinatorial Optimization and Its Application to Learning over Compressed Data

Takahiro Fujita

February, 2018

Abstract

Online prediction is a theoretical framework of sequential decision making such as weather forecasting and stock investment. The problem is formulated as a repeated game between the player and the adversarial environment, where in each trial the player first chooses a decision from a fixed decision space and then adversary returns a feedback. At this trial, the player incurs a loss defined by the decision and the feedback. The goal of the player is to minimize the total loss. The problem has been extensively studied for a couple of decades and still now it is a major topic in the machine learning community. In particular, for the online convex optimization, which is an online prediction problem where the decision space and loss functions are both restricted to be convex, several methodologies of designing and analyzing algorithms are established. Moreover, the methodologies have been recently applied to solving *offline* version of convex optimization of large scale, by utilizing the fact that the time and space complexity per trial of the online algorithms developed are typically very small. On the other hand, when the decision space is not convex, the problem is not fully understood.

Among non-convex problems, an especially important class is the online *combinatorial* optimization whose decision space is defined in some combinatorial way, since it has lots of real-world applications. The difficulty is that the corresponding offline combinatorial optimization problems are often NP-hard. In this study, we set two goals: One is to establish a general methodology of designing and analyzing *efficient* algorithms for online combinatorial optimization; and the other is to apply the methodology to the offline combinatorial optimization of large scale. This thesis makes some progress on the two goals and have the following three results.

First, we show that we have an efficient algorithm for online combinatorial optimization if the corresponding offline problem is approximable in polynomial time based on the LP relaxation. This result implies that we partially solved a long standing open problem on the *offline-online equivalence conjecture*. More precisely, we give a more general condition for the existence of an efficient algorithm for online combinatorial optimization: There exists a relaxed

convex space such that (i) the relaxed space is easy to project, and (ii) the relaxed space has an efficient *meta-rounding* algorithm. What we show is that this condition is satisfied when there exists an efficient approximation algorithm based on LP relaxation.

Second, we generalize the condition for the existence of an efficient algorithm for online combinatorial optimization. More precisely, we show that we can weaken the condition (i) to the one in terms of multiple subsequent projections. Informally, the condition is stated as follows: the relaxed space is easy to project from a further relaxed space, which is easy to project from an even further relaxed space, ... We then apply this general theory to a particular problem of online job scheduling.

Thirdly, we propose a new approach to large-scale machine learning, *learning over compressed data*: First compress the training data in a directed acyclic graph representation, and then employ a boosting algorithm over the compressed data, with the hope that the computation time is significantly reduced when the training data is well compressed. Here every instance in the training data corresponds to a path in the graph and thus the boosting can be viewed as a combinatorial optimization problem over the set of paths. Using a technique developed in the online linear optimization over paths, we give an algorithm that simulates the boosting and runs in time proportional to the size of the graph for each trial.

Acknowledgements

First and foremost, I would like to show my greatest appreciation to Professor Eiji Takimoto who provided carefully considered feedback and valuable comments. Special thanks also go to Associate Professor Kohei Hatano who provided technical help and sincere encouragement. I owe a very important debt to Associate Professor Shuji Kijima whose comments made enormous contribution to my work. I would like to thank Professor Masayuki Takeda whose opinions and information have helped me very much throughout the production of this study. I would also like to express my gratitude to my family for their moral support and warm encouragements.

The results in this thesis were published in proceedings of ALT 2013 [24], proceedings of ALT 2015 [22], and proceedings of WALCOM 2018 [25]. The journal version of ALT2015 will be published in The IEICE Transactions on Information and Systems [23]. This work is supported in part by JSPS KAKENHI Grant Number JP16J0462. Last but not least, I appreciate all editors, committees, anonymous referees, and publishers.

Contents

Chapter 1

Introduction

In this thesis, we consider the online prediction problem, which is a fundamental problem in the field of machine learning, and it has been studied extensively for decades. The problem is a repeated game between the player and the adversarial environment, where in each trial the player first chooses a decision from a fixed decision space and then adversary returns a feedback. At this trial, the player incurs a loss defined by the decision and the feedback. The goal of the player is to minimize the total loss. The feedback is not known in advance, and might be chosen by the adversary with full knowledge of the player's strategy. Many problems such as the web search engine and the recommendation system are modeled as online prediction problems. In particular, for the online convex optimization, which is an online prediction problem where the decision space and loss functions are both restricted to be convex, several methodologies of designing and analyzing algorithms are established [31, 10]. Moreover, the methodologies have been recently applied to solving *offline* version of convex optimization of large scale, by utilizing the fact that the time and space complexity per trial of the online algorithms developed are typically very small. On the other hand, when the decision space is not convex, the problem is not fully understood.

Among non-convex problems, an especially important class is the online *combinatorial* optimization whose decision space is defined in some combinatorial way, since it has lots of real-world applications [47, 13, 62, 64, 15, 41, 60, 59, 79, 42]. Examples of such decision spaces k-sets of a given universal set [78], s-t paths in a given graph [76], spanning trees of a given graph [11], permutations over a fixed item set [32, 83, 84], and so on. The difficulty is that the corresponding offline combinatorial optimization problems are often NP-hard and these decision spaces typically contain finite but exponentially many concepts. The amount of data is typically very large in these areas and even rapidly increasing. Therefore, it is particularly

important to develop the algorithms.

In this study, we set two goals: One is to establish a general methodology of designing and analyzing *efficient* algorithms for online combinatorial optimization; and the other is to apply the methodology to the offline combinatorial optimization of large scale.

1.1 Combinatorial Online Optimization

Background and motivation

We now consider the online combinatorial optimization problems. For example, consider the following network routing problem. Image that a network represented by a finite directed acyclic graphs. In each trial, we have to send along a stream of packets from distinguished vertex, called source, to another vertex, called destination. First, the system chooses a route connecting source and destination, and then send along a packet by the chosen route. The loss of system is the total delay on each edge composing the chosen route. In this problem, the number of routes is typically exponentially large in the number of edges.

The most well known problem in the online combinatorial problems is the expert problem. In each trial, the learner has to choose from the advice of n given expert (corresponding to decisions). The goal of the player is to do as the best expert in hindsight. In this case, we can employ a simple algorithm called the Hedge algorithm [19] (or Randomized Weighted Majority [48]), which keep track of a weight for each expert and makes predictions based on the weighted majority. However, when the concept classes contain finite but exponentially many concepts, this requires explicitly keeping track of a weight for each of the exponentially many combinatorial concepts, and thus results in an inefficient algorithm. Observe here that the decision space (i.e., the set of routes over graphs with n edge) is defined in a combinatorial way from a small seed, (i.e., the set of n edges), and the seed is only explicitly given to the player.

One of the major approaches to the combinatorial online optimization is to apply Follow the Regularized Leader (FTRL) framework [30], which is a general scheme of designing efficient combinatorial online algorithms with a good performance bound. In the framework, we assume that two external procedures called the projection and the decomposition are implemented. In other words, we need to design efficient algorithms for the two procedures individually for each concept class. In fact, in all the results mentioned above, algorithms for the two procedures are constructed individually for each of the concept classes.

Another approach to constructing online algorithm is to convert an offline approximation

algorithm for combinatorial optimization to online algorithm. More precisely, we assume an offline approximation algorithm for the linear optimization problem over the combinatorial concepts and the player has access to the offline algorithm as an oracle. The convert method called Follow the Perturbed Leader (FPL) [38] gives good loss bound, but the algorithm can use *exact* offline algorithm only. While the convert methods in [37, 26] can use *any* approximation offline algorithm and achieve an good loss bound, the computation complexity issue of the algorithms arise.

Our contributions

In this thesis, we propose a new construction method of efficient algorithms for the problems. The key notion of our result is metarounding, a robust version of rounding for relaxation-based approximation schema. Originally, metarounding was proposed by Carr and Vempala for approximately solving the multicast congestion problem [9]. We show that metarounding is quite suitable for online prediction of combinatorial concepts as well. Our algorithms consist of two components, an online prediction algorithm and a metarounding algorithm. Combined with a metarounding algorithm, FTRL and FPL achieve good regret bounds. One of our technical contribution is a new construction of metarounding using a *boosting* algorithms. Boosting [68] is a technique to construct highly accurate hypothesis by combining moderately accurate base hypotheses, which apparently seems nothing to do with metarounding. But, in fact, robust rounding and boosting hypotheses share a common structure and they are formulated as similar optimization problems. For this observation, we solve metarounding by using boosting-like algorithm. Our metarounding algorithms is adaptive in the sense that it does not require the explicit knowledge on the approximation ratio, which is advantageous in practice. Our preliminary experiments show that our metarounding algorithm indeed obtains better approximation ratios than theoretically guaranteed and its running time is much faster than a method based on previous work of Carr and Vempala.

In addition, for the FTRL + metarounding framework, we generalize the projection to *multiple projections*. As an application of the framework, we consider an online job scheduling problem with a single machine with precedence constraints under uncertainty. We formulate the problem as an online linear optimization problem over the permutahedron (the convex hull of permutation vectors) with specific linear constraints, in which the underlying decision space is written with exponentially many linear constraints. In this problem, we give polynomial time algorithms for the mulitiple projection and the metarounding. Our online algorithm pre-

dicts almost as well as the state-of-the-art offline approximation algorithms do in hindsight. In preliminary experiments, our algorithm runs considerably faster than the alternatives while performing competitively.

1.2 Learning over Compressed Data

Background and motivation

Most tasks in machine learning are formulated as optimization problems of various types. Recently, the amount of data to be treated is growing enormously large, and so the demands on scalable optimization methods are increasing. Probabilistic approach such as stochastic gradient descent methods [31] is now widely employed as standard techniques for large scale machine learning. Obviously, these methods require the time and/or the space complexity to be proportional to the size of given data.

On the other hand, the methodology of working over compressed data has gained much attention in the areas of database and data mining, where various methods have been developed, say, for the string search from a compressed string and the frequent word extraction from compressed texts [45, 33, 28]. But, as far as we are aware, most of all the methods developed so far are limited to simple tasks such as search and counting, and few results are known for more complex tasks such as optimization in machine learning.

Our contributions

In this thesis, we propose a new approach to large-scale machine learning, *learning over compressed data*: First compress the training data somehow and then employ various machine learning algorithms on the compressed data, with the hope that the computation time is significantly reduced when the training data is well compressed. As the first step, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data, which is a generalization of the ZDD by incorporating non-determinism. For the learning algorithm to be employed, we consider boosting algorithm called AdaBoost$*$ and its precursor AdaBoost. In this work, we give efficient implementations of the boosting algorithms whose running times (per iteration) are linear in the size of the given ZDD.

1.3 Organization

The rest of this thesis is organized as follows: In Chapter 2, we describe the online prediction model with basic learning algorithms and theory. In Chapter 3, we propose efficient algorithms for the online combinatorial optimization using matarounding. in Chapter 4, we describe a method for online scheduling with precedence constraints. In Chapter 5, we consider boosting over non-deterministic ZDDs. In Chapter 6, we give conclusion of this thesis.

Chapter 2

Online Combinatorial Optimization

In this chapter, we introduce the online combinatorial optimization, with some recent results on universal and efficient implementations of low-regret algorithmic frameworks such as Follow the Regularized Leader (FTRL) and Follow the Perturbed Leader (FPL).

2.1 Problem settings

We assume a finite class $\mathcal{C}$ of concepts, where each concept in $\mathcal{C}$ is encoded as a nonnegative vector in $\mathbb{R}^n$ for some integer n, i.e., $\mathcal{C} \subseteq \mathbb{R}^n$. The online prediction problem over a concept class $\mathcal{C}$ is described as a repeated game between the player and the adversary as follows: For each trial $t = 1, \ldots, T$

1. the player predicts $\boldsymbol{c}_t \in \mathcal{C}$
2. the adversary returns a loss vector $\boldsymbol{\ell}_t \in [0, 1]^n$
3. the player incurs loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.

The goal of the player is to minimize the α-regret: $\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t$. The first term represents the cumulative loss of the algorithm, and the second term the cumulative loss of the best concept in the class $\mathcal{C}$ in hindsight.

2.2 Combinatorial Concept Classes

In this section, we give the formal definition of combinatorial concept classes. A family of combinatorial concept classes $\mathcal{F}$ is defined by a pair $(\mathcal{S}, \phi)$, where $\mathcal{S}$ is a language over a finite

alphabet and $\phi : \mathcal{S} \to \mathcal{F}$ is a semantic function such that for each word $s \in \mathcal{S}$, $\phi(s)$ is a finite subset of $\mathbb{R}_+^n$ for some integer $n \geq 1$. Usually, n is upper bounded by a polynomial in $\|s\|$. In other words, a combinatorial concept $\mathcal{C}$ we consider in the thesis is $\phi(s)$ for some $s \in \mathcal{S}$ and semantic function ϕ. We call $\mathcal{S}$ a seed set. In this thesis, we consider an online algorithm that works efficiently and uniformly over the family $\mathcal{F}$. So, we assume that the algorithm receives a seed $s \in \mathcal{S}$ as input and makes predictions with the concept class $\mathcal{C} = \phi(s)$.

2.2.1 Examples

Below we show some examples of combinatorial concept classes.

Experts: The classical experts problem [19] is an example of our problem. Here the seed set $\mathcal{S}$ contains natural numbers, and for a seed n, the concept class $\phi(n)$ is the set $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n \subseteq \{0,1\}^n$, where $\boldsymbol{e}_i$ is a unit vector whose i-th component is 1 and other components are 0.

k-sets: The concept class of k-sets is a generalization of Experts, where each concept corresponds to a set of k experts among n experts. This problem was first studied by [78, 76]. Here the seed set $\mathcal{S}$ contains a pair (n, k) of natural numbers with $k \leq n$, and the concept class for a seed $s = (n, k)$ is $\phi(s) = \{\boldsymbol{c} \in \{0,1\}^n \| \sum_i^n c_i = k\}$.

s-t Paths: The online shortest path problem is studied in many papers such as [40]. Here, the seed set $\mathcal{S}$ contains directed graphs $G = (V, E)$ with two special nodes s and t, and the concept class for a seed G is $\phi(G) = \{c \in \{0,1\}^E \|$ the edge set $e \in E | c_e = 1$ forms an s-t paths in G $\}$. A related problem is also studied by [63].

Spanning Trees: The online problems for undirected or directed spanning trees are studied in [11, 32]. Here we consider undirected versions. Then, the seed set $\mathcal{S}$ contains undirected graphs $G = (V, E)$ and the concept class for a seed G is $\phi(G) = \{\boldsymbol{c} \in \{0,1\}^E \|$ the edge set $\{e \in E \| c_e = 1\}$ forms a spanning tree of $G\}$.

Permutations: Online linear optimization problem for this concept class models an online scheduling problem of n jobs with a single processor, where the sum of the flow time over all jobs is to be minimized [83]. Here the seed set $\mathcal{S}$ contains natural numbers and the concept class $\phi(n)$ for a seed n is the set of all permutations S_n over $\{1, \ldots, n\}$. A different representation of permutations and related problem is also studied by [32].

Rank Aggregation: The online rank aggregation problem is studied in [83], where the semantic function is different from the previous one. The problem is specified by the set of permutations S_n as follows: For each trial t, (i) the player predicts a permutation $\sigma_t \in S_n$, (ii)

the adversary returns a (true) permutation $\hat{\sigma}_t \in S_n$, and (iii) the player incurs the loss $d(\sigma_t, \hat{\sigma}_t)$, where d is the Kendall tau distance. For two permutations $\sigma_1, \sigma_2 \in S_n$, $d(\sigma_1, \sigma_2)$ is the number of index pairs (i, j) such that the relative order of $\sigma_1(i)$ and $\sigma_2(i)$ is inconsistent with that of $\sigma_1(j)$ and $\sigma_2(j)$. This is an online linear optimization problem with appropriate seed set S and semantic function ϕ. To see this, let the seed set S be the set of natural numbers, and let the concept class $\mathcal{C} = \phi(n)$ and the loss space L for a seed n be defined as $\mathcal{C} = L = \{\boldsymbol{c} \in \{0,1\}^n \| \boldsymbol{c}$ is a comparison vector $\}$. Here, a vector $\boldsymbol{c} \in \{0,1\}^{n^2}$ is a comparison vector if the set of index pairs $\{(i,j)|c_(i,j) = 1\}$ coincides with the set $\{(i,j)|\sigma(i) < \sigma(j)\}$ for some permutation $\sigma \in S_n$, which is denoted by $\sigma^{\boldsymbol{c}}$. Then, it is easy to see that for any permutations $\sigma^{\boldsymbol{c}}$ and $\sigma^{\boldsymbol{\ell}}$, $d(\sigma^{\boldsymbol{c}}, \sigma^{\boldsymbol{\ell}}) = \boldsymbol{c} \cdot \bar{\boldsymbol{\ell}}$, as required, where $\bar{\boldsymbol{\ell}}$ is a bit-wise complement of $.\boldsymbol{\ell}$

Set Covers: The problem is described by the following family $\mathcal{F} = (\mathcal{S}, \phi)$: The seed set S be a subset family U of some ground set $\mathcal{X}$ such that $\cup_{u \in U} u = \mathcal{X}$, and for a seed U, the concept class is $\phi(U) = \{\boldsymbol{c} \in \{0,1\}^U | \{u \in U | c_u = 1\}$ is a cover of $\mathcal{X}\}$.

Truth Assignments to CNF Formula: The problem is specied by a set of disjunctive clauses (which is a seed) $s = \{C_1, C_2, ..., C_m\}$ over k Boolean variables for some m and k, where each clause C_i is a disjunction of some literals. The online MAX-SAT problem for the seed s is the following: For each trial t, (i) the player predicts an assignment at $\boldsymbol{\ell} \in \{0,1\}^k$ to the variables, (ii) the adversary returns weights $\boldsymbol{a}_t \in [0,1]^m$ for the clauses, and (iii) the player gets reward dened by the sum of weight $\boldsymbol{\ell}_t, i$ over those clauses C_i that are satised by at. The problem is the reward version of an online linear optimization problem with the concept class $\mathcal{C} = \phi(s)$ and reward vector space L as described below. The class $\mathcal{C}$ consists of vectors in $\{0,1\}^n$ for $n = k+m$ such that the rst k bit vector a represents the truth assignment and the last m bit vector b represents the truth values of the clauses for the assignment a. That is, for each $1 \leq i \leq m$, $b_i = 1$ if and only if C_i is satised by $\boldsymbol{a}$. Note that the last m bits b are determined by the rst k bits $\boldsymbol{a}$. The reward space L consists of vectors $0^k\boldsymbol{\ell}$, where the rst k bits are 0 and $\boldsymbol{\ell} \in [0,1]^m$ represents the weights. So, the dot product of a concept $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$ and a reward 0^k becomes $\boldsymbol{b} \cdot \boldsymbol{\ell}$, which is the reward of the truth assignment $\boldsymbol{a}$ forthe weights $\boldsymbol{\ell}$, as required.

2.3 Follow the Regularized Leader

In this section, we introduce the online prediction model with a general scheme, called the Follow the Regularized Leader (FTRL) [30], of designing online prediction algorithms. Particularly in this thesis, we deal with the online prediction problem over combinatorial concept

classes, where the decision space is defined by some combinatorial way from a given seed.

2.3.1 Bregman Divergence

Let $\mathcal{X}$ be a subset of $\mathbb{R}^n$. Let $\Phi : \mathcal{X} \to \mathbb{R}$ be a continuously-differentiable strictly convex function. The Bregman Divergence Δ_Φ with respect to Φ (see, e.g., [10] for the details) is a function on pairs of points $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{X}$ where

$$\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = \Phi(\boldsymbol{p}) - \Phi(\boldsymbol{q}) - \nabla\Phi(\boldsymbol{q}) \cdot (\boldsymbol{p} - \boldsymbol{q})$$

By the strict convexity of Φ, the Bregman divergence Δ_Φ is always nonnegative and $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = 0$ if and only if $\boldsymbol{p} = \boldsymbol{q}$. We say that the function Φ is separable if Φ has the form $\Phi(\boldsymbol{p}) = \sum_{i=1}^n \phi_i(p_i)$ and Φ is uniformly separable if Φ is separable and $\phi_i = \phi$ for some ϕ. In our thesis, we will use functions Φ which are uniformly separable. For example, for $\mathcal{X} \subset \mathbb{R}^n$ and $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$, the corresponding Bregman divergence $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q})$is $\frac{1}{2}\|\boldsymbol{p} - \boldsymbol{q}\|_2^2$ and for $\mathcal{X} = \{\boldsymbol{x} \in [0,1]^n \mid \sum_{i=1}^n x_i = 1\}$ and $\Phi(\boldsymbol{x}) = \sum_{i=1}^n -x_i \ln x_i$, $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^n p_i \ln(p_i/q_i)$, which is the Kullback-Leibler divergence or the relative entropy between distributions. We will use a geometric property of the Bregman divergence which is known as the Generalized Pythagorean Theorem.

Theorem 1 (Generalized Pythagorean Theorem (e.g., [8, 10]))**.** *Let $\mathcal{X}' \subset \mathcal{X}$ be any closed convex set. Let $\boldsymbol{q}$ be any point in $\mathcal{X}$ and let $\boldsymbol{p} = \inf_{\boldsymbol{p}' \in \mathcal{X}'} \Delta_\Phi(\boldsymbol{p}', \boldsymbol{q})$. Then, it holds for any $\boldsymbol{r} \in \mathcal{X}'$ that*

$$\Delta_\Phi(\boldsymbol{r}, \boldsymbol{q}) \geq \Delta_\Phi(\boldsymbol{r}, \boldsymbol{p}) + \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}).$$

In particular, the equality holds when $\mathcal{X}'$ is an affine set.

2.3.2 Algorithm

We first consider the online linear optimization problem over a convex decision space $\mathcal{P} \subseteq \mathbb{R}^n$. The FTRL needs an external procedure Projection.

Definition 1. *The Projection onto a convex set $\mathcal{P}$ with respect to a Bregman divergence Δ_Φ is a procedure that takes as input a point $z \in \mathcal{X}$ and outputs* $\arg\inf_{x \in \mathcal{P}} \Delta_\Phi(x, z)$.

At each trial t, the FTRL maintains a point $x_t \in \mathcal{P}$. Given a loss vector t from the adversary, it updates x_t to $x_{t+1/2}$ by solving a regularized convex optimization. Finally, it runs the

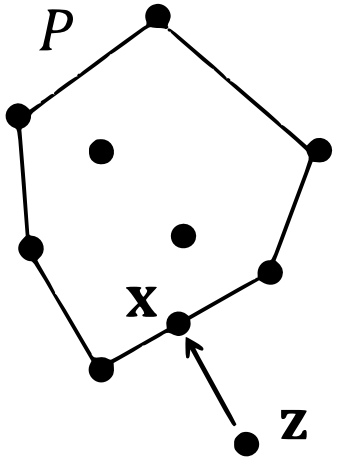

Figure 2.1: projection

procedure Projection and get x_{t+1}, which is the projection of $x_{t+1/2}$ onto P with respect to the Bregman divergence Δ_Φ. See Algorithm 1 for more details.

Algorithm 1 Follow the Regularized Leader

Input: parameter $\eta > 0$, a separable strictly convex function Φ.

1. Let $\boldsymbol{x}_1$ be any point in $\mathcal{P}$.
2. For $t = 1, \ldots, T$
   (a) Predict $\boldsymbol{x}_t \in \mathcal{P}$
   (b) Incur a loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.
   (c) Update $\boldsymbol{x}_{t+1/2} = \arg\min_{\boldsymbol{x} \in X} \eta \boldsymbol{x} \sum_{j=1}^{t} \boldsymbol{\ell}_j + \Phi(\boldsymbol{x})$.
   (d) (Projection) $\boldsymbol{x}_{t+1}$ =Projection($\boldsymbol{x}_{t+1/2}$).

The following proposition is known.

Lemma 1. *For every $\boldsymbol{u} \in \mathcal{P}$, the algorithm enjoys the following regret guarantee*

$$\sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{u}) \cdot \boldsymbol{\ell}_t \leq \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{x}_{t-1}) \cdot \boldsymbol{\ell}_t + \frac{1}{\eta} (\Phi(\boldsymbol{u}) - \Phi(\boldsymbol{x}_1))$$

Proof. The lemma is proved by induction on T. Assume that for T, we have

$$\sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{u}) \cdot \boldsymbol{\ell}_t \leq \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{x}_{t+1}) \cdot \boldsymbol{\ell}_t$$

and let us prove for $T+1$. Since $\boldsymbol{x}_{T+1} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=0}^{T+1} \boldsymbol{x}\cdot\boldsymbol{\ell}_t$ we have:

$$\begin{aligned}
\sum_{t=0}^{T+1} \boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \sum_{t=0}^{T+1} \boldsymbol{u}\cdot\boldsymbol{\ell}_t &\le \sum_{t=0}^{T+1} \boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \sum_{t=0}^{T+1} \boldsymbol{x}_{T+2}\cdot\boldsymbol{\ell}_t \\
&= \sum_{t=0}^{T} (\boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \boldsymbol{x}_{T+2}\cdot\boldsymbol{\ell}_t) + \boldsymbol{x}_{T+1}\cdot\boldsymbol{\ell}_{T+1} - \boldsymbol{x}_{T+2}\cdot\boldsymbol{\ell}_{T+1} \\
&\le \sum_{t=0}^{T} (\boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \boldsymbol{x}_{t+1}\cdot\boldsymbol{\ell}_t) + \boldsymbol{x}_{T+1}\cdot\boldsymbol{\ell}_{T+1} - \boldsymbol{x}_{T+2}\cdot\boldsymbol{\ell}_{T+1} \\
&= \sum_{t=0}^{T+1} (\boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \boldsymbol{x}_{t+1}\cdot\boldsymbol{\ell}_t)
\end{aligned}$$

where in the first line we used the induction hypothesis for $\boldsymbol{u} = \boldsymbol{x}_{T+2}$. We conclude that

$$\begin{aligned}
\sum_{t=1}^{T} \boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \sum_{t=1}^{T} \boldsymbol{u}\cdot\boldsymbol{\ell}_t &\le \sum_{t=1}^{T} \boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \sum_{t=1}^{T} \boldsymbol{x}_{t+1}\cdot\boldsymbol{\ell}_t - \boldsymbol{x}_0\cdot\boldsymbol{\ell}_0 + \boldsymbol{u}\cdot\boldsymbol{\ell}_0 + \boldsymbol{x}_0\cdot\boldsymbol{\ell}_0 - \boldsymbol{x}_1\cdot\boldsymbol{\ell}_0 \\
&= \sum_{t=1}^{T} \boldsymbol{x}_t\cdot\boldsymbol{\ell}_t - \sum_{t=1}^{T} \boldsymbol{x}_{t+1}\cdot\boldsymbol{\ell}_t + \frac{1}{\eta}(\Phi(\boldsymbol{u}) - \Phi(\boldsymbol{x}_1))
\end{aligned}$$

□

Theorem 2. *For an appropriate choice of η, the expected 1-regret of the FTRL is $O(\sqrt{\lambda DT})$ where $D = \max_{\boldsymbol{c},\boldsymbol{c}'\in\mathcal{C}} \|\boldsymbol{c}-\boldsymbol{c}'\|_2$.*

Proof. Now recall that $\Phi(\boldsymbol{x})$ is a convex function and $\mathcal{P}$ is convex. Then by Taylor expansion at $\boldsymbol{x}_{t+1}$, there exists a $\boldsymbol{z}_t \in [\boldsymbol{x}_t, \boldsymbol{x}_{t+1}]$ for which

$$\begin{aligned}
\Phi_t(\boldsymbol{x}_t) &= \Phi_t(\boldsymbol{x}_{t+1}) + (\boldsymbol{x}_t - \boldsymbol{x}_{t+1})\cdot\nabla\Phi_t(\boldsymbol{x}_{t+1}) + \frac{1}{2}\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|^2 \\
&\ge \Phi_t(\boldsymbol{x}_{t+1}) + \frac{1}{2}\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|^2
\end{aligned}$$

Recall our notation $\|\boldsymbol{y}\|^2 = \boldsymbol{y}\nabla^2\Phi_t(\boldsymbol{z})\boldsymbol{y}$ and it follows that $\|\boldsymbol{y}\|^2 = \boldsymbol{y}\nabla^2\Phi_t(\boldsymbol{z})\boldsymbol{y}$. The inequality above is true because $\boldsymbol{x}_t$ is minimum of Φ_t over $\mathcal{P}$. Thus,

$$\begin{aligned}
\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|^2 &\le 2\Phi_t(\boldsymbol{x}_t) - 2\Phi_t(\boldsymbol{x}_{t+1}) \\
&= 2(\Phi_{t-1}(\boldsymbol{x}_t) + \Phi_{t-1}(\boldsymbol{x}_{t+1}) + 2\eta(\boldsymbol{x}_t - \boldsymbol{x}_{t+1})\cdot\boldsymbol{\ell}_t \\
&\le 2\eta(\boldsymbol{x}_t - \boldsymbol{x}_{t+1})\cdot\boldsymbol{\ell}_t
\end{aligned}$$

By the generalized Cauchy-Schwartz inequality,

$$\begin{aligned}(\boldsymbol{x}_t - \boldsymbol{x}_{t+1}) \cdot \boldsymbol{\ell}_t &\leq \|\boldsymbol{\ell}_t\|^* \cdot \|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\| \\ &\leq \|\boldsymbol{\ell}_t\|^* \cdot \sqrt{2\eta(\boldsymbol{x}_t - \boldsymbol{x}_{t+1}) \cdot \boldsymbol{\ell}_t}\end{aligned}$$

Shifting sides and squaring we get

$$(\boldsymbol{x}_t - \boldsymbol{x}_{t+1}) \cdot \boldsymbol{\ell}_t \leq 2\eta \|\boldsymbol{\ell}_t\|^{*2}$$

This together with the Lemma, summing over T periods we obtain

$$\sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{u}) \cdot \boldsymbol{\ell}_t \leq 2\eta\lambda T + \frac{1}{\eta}[\Phi(\boldsymbol{u}) - \Phi(\boldsymbol{x}_1)]$$

Choosing the optimal η, we obtain the Theorem. □

Proposition 3. *Let $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$. Then, for an appropriate choice of η, the expected 1-regret of the FTRL is $O(D\sqrt{nT})$ where $D = \max_{\boldsymbol{c},\boldsymbol{c}' \in \mathcal{C}} \|\boldsymbol{c} - \boldsymbol{c}'\|_2$.*

Proposition 4. *Let $\Phi(\boldsymbol{x}) = \sum_{i=1}^{n} -x_i \ln x_i$. Then, for an appropriate choice of η, the expected 1-regret of the FTRL is $O(\sqrt{L^* D} + D)$ where $D = \max_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t$.*

2.3.3 FTRL over a Combinatorial Concept Class

Next we consider the online linear optimization problem over a combinatorial concept class C. In this case, we let $\mathcal{P}$ be the convex hull of $\mathcal{C}$ and run Algorithm 1 to get predictions $\boldsymbol{x}_t \in \mathcal{P}$ for all trials t. To get predictions from the concept class $\mathcal{C}$, we need an external procedure Decomposition.

Definition 2. *The Decomposition for a concept class $\mathcal{C}$ is a randomized procedure that takes as input a point $\boldsymbol{x}$ in the convex hull of $\mathcal{C}$ and outputs a point $\mathbf{c} \in \mathcal{C}$ randomly so that $E[\mathbf{c}] = \boldsymbol{x}$.*

So the FTRL for $\mathcal{C}$ is summarized in Algorithm 2. Since $E[\mathbf{c}_t \cdot \boldsymbol{\ell}_t] = \boldsymbol{x}_t \cdot \boldsymbol{\ell}_t$ for all t, we have the following regret bound.

Corollary 5. *Let λ, D and η be the parameters defined in Proposition 2. Then the expected 1-regret of FTRL of Algorithm 2 is at most $2\sqrt{2\lambda DT}$.*

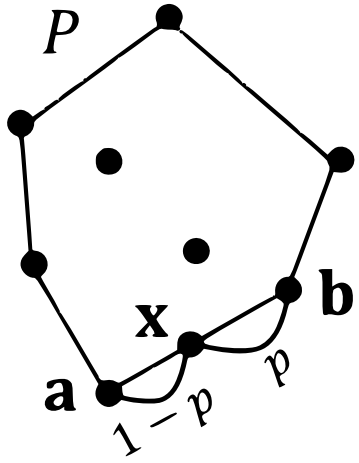

Figure 2.2: decomposition

Algorithm 2 FTRL over a combinatorial concept class $\mathcal{C}$

1. For $t = 1, \ldots, T$
   (a) Run Algprithm 1 one step $\boldsymbol{x}_t \in \mathcal{P}$
   (b) Run Decomposition($\boldsymbol{x}_t$) and get $\boldsymbol{c}_t \in \mathcal{C}$ randomly so that $E[\boldsymbol{c}_t] = \boldsymbol{x}_t$.
   (c) Predict $\boldsymbol{c}_t$and incur a loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.
   (d) Feed $\boldsymbol{\ell}_t$ to Algorithm 1 and resume it.

2.4 Follow the Perturbed Leader

A generic alternative approach for combinatorial online prediction is to convert offline approximation algorithms for combinatorial optimization to online prediction algorithms. More precisely, we assume an offline α-approximation algorithm A for the linear optimization problem $OPT(\mathcal{C})$. The algorithm is supposed to take a loss vector $\ell \in [0,1]^n$ as input and outputs $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{c}' \in \mathcal{C}} \boldsymbol{c}' \cdot \boldsymbol{\ell}$. We assume that the player can use the offline algorithm A as an oracle, and each call of the oracle can be done in a unit time.

The goal of the player is to minimize the α-regret:

$$\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t.$$

The α-regret measures the difference between cumulative loss of the player and that of an α-approximate fixed concept in hindsight, which can be computed by the approximation algorithm

A.

There are three main previous researches on the player's strategy for combinatorial online prediction problems where approximation algorithms are available.

Kalai and Vempala proposed Follow the perturbed leader (FPL [38]). FPL is a simple algorithm that adds random perturbations to the cumulative loss of each component, and then predicts with the combinatorial object that has the minimum perturbed loss. FPL uses an exact offline optimization algorithm (i.e., $\alpha = 1$).

Algorithm 3 Follow the Perturbed Leader

Input: parameter $\epsilon > 0$.

1. For $t = 1, \ldots, T$
   (a) Choose $\boldsymbol{p}_t$ uniformly at random from the cube $[0, \frac{1}{\epsilon}]^n$
   (b) use $A(\boldsymbol{\ell}_1 + \boldsymbol{\ell}_2 + \ldots \boldsymbol{\ell}_{t-1} + \boldsymbol{p}_t)$.

Theorem 6. *For an appropriate choice of ϵ, the expected 1-regret of the FPL is $O(\sqrt{n^2 DT})$ where $D = \max_{\boldsymbol{c},\boldsymbol{c}' \in \mathcal{C}} \|\boldsymbol{c} - \boldsymbol{c}'\|_1$.*

Proof. First, we see by induction on T that using $A(\sum_{i=1}^{t} \boldsymbol{\ell}_i)$ on trial t gives 0 regret,

$$\sum_{t=1}^{T} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i) \cdot \boldsymbol{\ell}_t \leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t$$

For $T = 1$, it is trivial. For the induction step from T to $T + 1$,

$$\begin{aligned}
\sum_{t=1}^{T+1} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i) \cdot \boldsymbol{\ell}_t &= \sum_{t=1}^{T} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i) \cdot \boldsymbol{\ell}_t + A(\sum_{t=1}^{T+1} \boldsymbol{\ell}_t) \cdot \boldsymbol{\ell}_{T+1} \\
&\leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + A(\sum_{t=1}^{T+1} \boldsymbol{\ell}_t) \cdot \boldsymbol{\ell}_{T+1} \\
&\leq A(\sum_{i=1}^{T+1} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + A(\sum_{t=1}^{T+1} \boldsymbol{\ell}_t) \cdot \boldsymbol{\ell}_{T+1} \\
&= A(\sum_{i=1}^{T+1} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T+1} \boldsymbol{\ell}_t
\end{aligned}$$

Where in the third line we used the definition of algorithm A. Then, by using this inequality,

$$\begin{aligned}\sum_{t=1}^{T+1} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i + \boldsymbol{p}_t) \cdot (\boldsymbol{\ell}_t + \boldsymbol{p}_t - \boldsymbol{p}_{t-1}) &\leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t + \boldsymbol{p}_T) \cdot (\sum_{t=1}^{T} \boldsymbol{\ell}_t + \boldsymbol{p}_T) \\ &\leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot (\sum_{t=1}^{T} \boldsymbol{\ell}_t + \boldsymbol{p}_T) \\ &= A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + \sum_{t=1}^{T} (A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot (\boldsymbol{p}_t - \boldsymbol{p}_{t-1})).\end{aligned}$$

Rearranging we have that

$$\begin{aligned}&\sum_{t=1}^{T+1} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i + \boldsymbol{p}_t) \cdot \boldsymbol{\ell}_t \\ &= A(\sum_{i=1}^{T} \boldsymbol{\ell}_i) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + \sum_{t=1}^{T} (A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) - A(\sum_{t=1}^{T} \boldsymbol{\ell}_t + \boldsymbol{p}_t)) \cdot (\boldsymbol{p}_t - \boldsymbol{p}_{t-1}) \\ &\leq A(\sum_{i=1}^{T} \boldsymbol{\ell}_i) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + D \sum_{t=1}^{T} \|\boldsymbol{p}_t - \boldsymbol{p}_{t-1}\|_\infty\end{aligned}$$

where $D = \max_{\boldsymbol{c}, \boldsymbol{c}' \in \mathcal{C}} \|\boldsymbol{c} - \boldsymbol{c}'\|_1$ and $\boldsymbol{x} \cdot \boldsymbol{y} \leq \|\boldsymbol{x}\|_1 \|\boldsymbol{y}\|_\infty$. In terms of expected performance, it wouldn't matter whether we chose a new $\boldsymbol{p}_t$ each trial or whether $\boldsymbol{p}_t = \boldsymbol{p}_1$ for all $t > 1$. Thus,

$$\begin{aligned}\sum_{t=1}^{T+1} A(\sum_{i=1}^{t} \boldsymbol{\ell}_i + \boldsymbol{p}_1) \cdot \boldsymbol{\ell}_t &\leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + D\|\boldsymbol{p}_1\|_\infty \\ &\leq A(\sum_{t=1}^{T} \boldsymbol{\ell}_t) \cdot \sum_{t=1}^{T} \boldsymbol{\ell}_t + \frac{D}{\epsilon}.\end{aligned}$$

Take a random point $\boldsymbol{x} \in [0, \frac{1}{\epsilon}]^n$. If $\boldsymbol{x} \notin [0, \frac{1}{\epsilon}]^n$, then for some $i, x_i \notin v_i + [0, \frac{1}{\epsilon}]$, which happens with probability at most $\epsilon \|v_i\|$ for any particular i. By the union bound, for any $\boldsymbol{v} \in \mathbb{R}^n$, the cube $[0, \frac{1}{\epsilon}]$ and $\boldsymbol{v} + [0, \frac{1}{\epsilon}]$ overlap in at least a $(1 - \epsilon\|\boldsymbol{v}\|_1)$ fraction. Thus,

$$E[A(\sum_{i=1}^{t-1} \boldsymbol{\ell}_i + \boldsymbol{p}_t) \cdot \boldsymbol{\ell}_t] \leq E[A(\sum_{i=1}^{t} \boldsymbol{\ell}_i + \boldsymbol{p}_t) \cdot \boldsymbol{\ell}_t] + (1 - \epsilon\|\boldsymbol{\ell}_t\|_1)\|\boldsymbol{c}\|_2$$

Rearranging, summing over all T iterations, we have that

$$\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t \leq \epsilon RT + \frac{D}{\epsilon}$$

Choosing the optimal ϵ, we obtain the Theorem. □

Chapter 3

Combinatorial Online Prediction via Metarounding

3.1 Introduction

Online prediction problems of combinatorial concepts arise in many situations such as routing, ranking and scheduling. Examples of such combinatorial concepts include, s-t paths, set covers, permutations and so on. In a combinatorial online prediction problem, we assume a finite set $\mathcal{C} \subseteq \mathbb{R}^n$ of combinatorial concepts, where each combinatorial concept is represented as a vector in $\mathbb{R}^n$. Then we consider the following protocol between the player and the adversary: For each trial $t = 1, ..., T$, (i) the player predicts $\boldsymbol{c}_t \in \mathcal{C}$, (ii) the adversary returns a loss vector $\boldsymbol{\ell}_t \in \mathcal{L} \subseteq [0, 1]^n$, and (iii) the player incurs loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$. Typical goal of the player is to minimize the regret $\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t$.

A straightforward approach for combinatorial online prediction problems is to apply Hedge [19]. Given a set of experts (i.e., prediction strategies or algorithms), Hedge is guaranteed to predict almost as well as the best expert in hindsight. So, with each combinatorial concept $\boldsymbol{c} \in \mathcal{C}$ as an expert, Hedge can achieve a good regret bound [1]. This approach, however, is inefficient in general since there are exponentially many concepts in the class $\mathcal{C}$ and thus Hedge takes exponential time at each trial.

There are many results on efficient combinatorial online prediction algorithms for individual concepts such as k-sets [78], permutations [32, 83, 84], spanning trees [11] and so on. There are some work on classes of combinatorial concepts [40, 11, 74].

A generic alternative approach for combinatorial online prediction is to convert offline ap-

[1] Note that Hedge is suboptimal in some cases (see [6] for the details).

proximation algorithms for combinatorial optimization to online prediction algorithms. More precisely, we assume an offline α-approximation algorithm A for the linear optimization problem over $\mathcal{C}$. The algorithm is supposed to take a loss vector $\ell \in [0,1]^n$ as input and outputs $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{c}' \in \mathcal{C}} \boldsymbol{c}' \cdot \boldsymbol{\ell}$. We assume that the player can use the offline algorithm A as an oracle, and each call of the oracle can be done in a unit time.

The goal of the player is to minimize the α-regret:

$$\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t.$$

The α-regret measures the difference between cumulative loss of the player and that of an α-approximate fixed concept in hindsight, which can be computed by the approximation algorithm A.

There are two main previous researches on the player's strategy for combinatorial online prediction problems where approximation algorithms are available. First, Kalai and Vempala proposed Follow the perturbed leader (FPL [38]). FPL uses an exact offline optimization algorithm (i.e., $\alpha = 1$). The 1-regret bound of FPL is $O(\sqrt{T})$ and its running time per trial is $O(n)$. FPL also works with α-approximation algorithms, however, its α-regret bound becomes $O(\alpha^T \sqrt{T})$ in general.

Kakade et al. proposed a different strategy using α-approximation algorithms, which achieves $O(\alpha\sqrt{T})$ α-regret bound [37]. The running time of the algorithm is $O(poly(n)T)$. Unfortunately, the time complexity at each trial depends on T, which is not desirable in practice.

In this thesis, we consider a slightly stronger assumption on the offline approximation algorithms. We assume that the player has access to the following approximation algorithm A:

Assumption 1. *Given $\boldsymbol{\ell} \in [0,1]^n$ as input, A outputs $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{x} \cdot \boldsymbol{\ell}$ for some $\alpha > 1$, where $\mathcal{P}$ is a convex set such that $\mathcal{P} \supset \mathcal{C}$, and linear optimization over $\mathcal{P}$ can be done in polynomial time in n.*

This assumption is motivated by the fact that many combinatorial optimization problems have LP relaxation schemes. Then, our main result is stated as follows:

Theorem 7. *Under Assumption 1, there exists a strategy of the player whose α-regret bound is $O((\alpha + \varepsilon)\sqrt{T})$ and its running time is polynomial in n and $1/\varepsilon$ for any $\varepsilon > 0$.*

The key notion of our result is *metarounding*, a robust version of rounding for relaxation-based approximation schema. Originally, metarounding was proposed by Carr and Vempala for

approximately solving the multicast congestion problem [9]. We will show that metarounding is quite suitable for online prediction of combinatorial concepts as well.

One of our technical contribution is a new construction of metarounding using a *boosting* algorithms. Boosting [68] is a technique to construct highly accurate hypothesis by combining moderately accurate base hypotheses, which apparently seems nothing to do with metarounding. But, in fact, robustifying rounding and boosting hypotheses share a common structure and they are formulated as similar optimization problems.

Our algorithm is adaptive in the sense that it does not require the explicit knowledge on the approximation ratio α, which is advantageous in practice. Our preliminary experiments show that our algorithm indeed obtains better approximation ratios than theoretically guaranteed and its running time is much faster than a method based on previous work of Carr and Vempala.

3.2 Preliminaries

Let $\Phi : \Gamma \to \mathbb{R}$ be a strictly convex function defined on a closed convex set $\Gamma \subseteq \mathbb{R}^n$. The Bregman divergence Δ_Φ with respect to Φ is defined as $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = \Phi(\boldsymbol{p}) - \Phi(\boldsymbol{q}) - \nabla\Phi(\boldsymbol{q}) \cdot (\boldsymbol{p} - \boldsymbol{q})$.

The *unnormalized relative entropy* $\Delta(\boldsymbol{p}, \boldsymbol{q})$ from $\boldsymbol{q} \in \mathbb{R}^n_+$ to $\boldsymbol{p} \in \mathbb{R}^n_+$ is defined as

$$\Delta(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{n} p_i \ln \frac{p_i}{q_i} + \sum_{i=1}^{n} q_i - \sum_{i=1}^{n} p_i.$$

It is known that $\Delta(\boldsymbol{p}, \boldsymbol{q}) \geq 0$ and $\Delta(\boldsymbol{p}, \boldsymbol{q}) = 0$ if and only if $\boldsymbol{p} = \boldsymbol{q}$. Unnormalized relative entropy is not symmetric in general, i.e., $\Delta(\boldsymbol{p}, \boldsymbol{q}) \neq \Delta(\boldsymbol{q}, \boldsymbol{p})$ for some $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^n_+$. In fact, unnormalized relative entropy is a special case of Bregman divergence [10].

3.3 Main Structure

In this section, we describe the main structure of our algorithms for combinatorial online prediction. Let $\mathcal{P} \subset \mathbb{R}^n_+$ such that (i) $\mathcal{P}$ is convex, (ii) $\mathcal{P}$ contains $\mathcal{C}$, and (iii) linear optimization over $\mathcal{P}$ can be done in polynomial time in n. For example, $\mathcal{P}$ might be represented a set of $poly(n)$ linear constraints. Then linear programming over $\mathcal{P}$ can be solved in polynomial time, say, by using interior point methods. Also, $\mathcal{P}$ might be described as linear constraints and semidefinite constraints. Then linear optimization over $\mathcal{P}$ belongs to semidefinite programs, which are

solvable efficiently.

Our algorithms consist of two components, an online prediction algorithm and a metarounding algorithm. At each trial t, our online prediction algorithm predict $\boldsymbol{x}_t$ not in $\mathcal{C}$, but in a "relaxed"space $\mathcal{P}$. Then the metarounding algorithm chooses $\boldsymbol{c}_t \in \mathcal{C}$ by "rounding" $\boldsymbol{x}_t$, where, roughly speaking, $\boldsymbol{c}_t$ is close to $\boldsymbol{x}_t$. Then, the regret of the algorithm would be close to that of online prediction algorithms over $\mathcal{P}$. We give a formal definition of metarounding algorithms as follows.

Definition 3. *An algorithm A is a metarounding algorithm if A is given $\boldsymbol{x} \in \mathcal{P}$ as input and outputs $\boldsymbol{c} \in \mathcal{C}$ such that for any $\boldsymbol{\ell} \in \mathcal{L} \subseteq [0,1]^n$, $E[\boldsymbol{c} \cdot \boldsymbol{\ell}] \leq \alpha \boldsymbol{x} \cdot \boldsymbol{\ell}$, where the expectation is taken w.r.t. the internal randomness of the algorithm A.*

The notion of metarounding was first proposed by Carr and Vempala [9]. Given an α-approximation algorithm with relaxation, which requires $\boldsymbol{\ell} \in \mathcal{L} = [0,1]^n$ and outputs $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{x} \cdot \boldsymbol{\ell}$, they show how to construct a metarounding algorithm by using the approximation algorithm as an oracle. We will propose more efficient methods to construct a metarounding algorithm from the approximation algorithm.

A related notion is proposed by Kalai and Vempala [38]. An algorithm has α-pointwise approximation property if, given any $\boldsymbol{\ell} \in \mathcal{L}$, the algorithm outputs $\boldsymbol{c} \in \mathcal{C}$ such that $c_i \leq \alpha c_i^*$ for $i = 1, \ldots, n$, where $\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}' \in \mathcal{C}} \boldsymbol{c}' \cdot \boldsymbol{\ell}$. Kalai and Vempala showed that FPL with approximation algorithm with this property can achieve good α-regret bounds [38]. In particular, when $\mathcal{L} = [0,1]^n$, the notion of metarounding turns out to be equivalent with α-pointwise approximation property in some sense (shown in Proposition 4). But, in general cases where $\mathcal{L} \subset [0,1]^n$, both notions seem to be incomparable with each other. Further, as we will show, the notion of metarounding is applicable more widely than the point-wise approximation property.

We show that any online prediction algorithm A whose prediction space is $\mathcal{P}$ can be combined with a metarounding algorithm. At each trial t, the combined algorithm gets the prediction $\boldsymbol{p}_t$ of the precition algorithm A, gives the prediction $\boldsymbol{p}_t$ to the metarounding algorithm as an input, and get the combinatrial concept $\boldsymbol{c}_t$, which is used as the prediction of the combined algorithm.

Proposition 8. *Suppose that there exists an online prediction algorithm A whose prediction $\boldsymbol{p}_t$ at each trial t belongs to $\mathcal{P}$ and its 1-regret w.r.t. $\mathcal{P}$ is bounded by Reg_A. Then, the α-regret of A combined with a metarounding algorithm w.r.t. the concept class $\mathcal{C}$ is at most αReg_A.*

Proof. Assume that the algorithm A with a metarounding algorithm predicts $\boldsymbol{c}_t$ at each trial t. Then,

$$\begin{aligned}\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t &\leq \alpha \sum_{t=1}^{T} \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t \text{ (by definition of metarounding)} \\ &\leq \alpha \min_{\boldsymbol{p} \in \mathcal{P}} \sum_{t=1}^{T} \boldsymbol{p} \cdot \boldsymbol{\ell}_t + \alpha Reg_A \text{ (by definition of algorithm } A) \\ &\leq \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t + \alpha Reg_A \text{ (since } \mathcal{C} \subseteq \mathcal{P}).\end{aligned}$$

□

As corollaries, combined with a metarounding algorithm, Follow the Regularized Leader(FTRL, e.g., [30]) and FPL achieve good α-regret bounds. In particular, FTRL generalizes popular algorithms such as Hedge [19] and OGD [86], respectively. The details of the algorithms are shown in Algorithm 5 and 4, respectively.

Corollary 9. *Let $\lambda = \max_{t, \boldsymbol{x} \in \mathcal{P}} \boldsymbol{\ell}_t^T [\nabla^2 \Phi(\boldsymbol{x})]^{-1} \boldsymbol{\ell}_t$ and $D = \max_{\boldsymbol{x} \in \mathcal{P}} \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1)$. Then, with an appropriate choice of η, the α-regret of FTRL(η) with Metarounding is $O(\alpha\sqrt{\lambda D T})$.*

Corollary 10. *Let $D \geq |\boldsymbol{c} - \boldsymbol{c}'|_1$ for all $\boldsymbol{c}, \boldsymbol{c}' \in C$. Then, with an appropriate choice of η, the α-regret of FPL(η) with Metarounding is $O(\alpha D\sqrt{nT})$.*

Algorithm 4 FTRL(η) with Metarounding

1. Let $\boldsymbol{x}_1 \in \mathcal{P}$ be any initial point.
2. For $t = 1, \ldots, T$
   (a) Run the metarounding with $\boldsymbol{x}_t$ and get $\boldsymbol{c}_t \in \mathcal{C}$.
   (b) Predict $\boldsymbol{c}_t$ and incur loss $\boldsymbol{c} \cdot \boldsymbol{\ell}_t$.
   (c) Update $\boldsymbol{x}_{t+1/2} = \arg\min_{\boldsymbol{x} \in \mathcal{P}} \eta \boldsymbol{x} \cdot \sum_{j=1}^{t} \boldsymbol{\ell}_j + \Phi(\boldsymbol{x})$.
   (d) (Projection) Let $\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x} \in \mathcal{P}} \Delta_\Phi(\boldsymbol{x}, \boldsymbol{x}_{t+1/2})$.

3.4 Examples and Applications of Metarounding

We show some examples and applications of metarounding.

Algorithm 5 FPL(η) with Metarounding

1. Let $\boldsymbol{\ell}_0 = \mathbf{0}$.
2. For $t = 1, \ldots, T$
   (a) Solve the linear program over $\mathcal{P}$: $\boldsymbol{x}_t = \arg\min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{x} \cdot \left(\sum_{j=0}^{t-1} \boldsymbol{\ell}_j + \boldsymbol{p}_t \right)$, where $\boldsymbol{p}_t$ is a uniform random vector in $[0, 1/\eta]^n$.
   (b) Run the metarounding algorithm with $\boldsymbol{x}_t$ and get $\boldsymbol{c}_t \in \mathcal{C}$.
   (c) Predict $\boldsymbol{c}_t \in \mathcal{C}$ and incur the loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.

3.4.1 Online Set Cover

Let S be a finite set and $U \subseteq 2^S$ be a fixed set of the subsets of S. A cover is a subset U' of U that satisfies $\bigcup_{u \in U'} u = S$. The online set cover problem is stated as follows: For each trial t, (i) the player predicts a cover $U_t \subseteq U$, (ii) the adversary returns weights $\boldsymbol{\ell}_t \in [0, 1]^U$, and (iii) the player incurs loss $\sum_{u \in U_t} \ell_t(u)$. The problem is a combinatorial online prediction problem with the concept class $\mathcal{C} = \{\boldsymbol{c} \in \{0, 1\}^U \mid \{u \in U \mid c(u) = 1\} \text{ is a cover}\}$ and the loss vector space $\mathcal{L} = [0, 1]^U$.

The corresponding offline optimization problem is the weighted minimum set cover problem. The problem has an $O(\log |S|)$ approximation algorithm using an LP-relaxation and a randomized metarounding [73]. The relaxed space $\mathcal{P} \subseteq [0, 1]^U$ is described as the set of feasible solutions $\boldsymbol{x} \in [0, 1]^U$ that satisfies the following linear constraints: $\sum_{u \in U : s \in u} x(u) \geq 1$ ($\forall s \in S$). The metarounding is simple: Given a feasible solution $\boldsymbol{x} \in \mathcal{P}$ as input, set $c(u) = 1$ with probability $x(u)$ and $c(u) = 0$ with probability $1 - x(u)$. It is shown that the metarounding has approximation factor $O(\log |S|)$. That is, for any loss vector $\boldsymbol{\ell} \in [0, 1]^U$, it holds that $E[\boldsymbol{c} \cdot \boldsymbol{\ell}] = O(\log |S|) \boldsymbol{x} \cdot \boldsymbol{\ell}$.

3.4.2 Online MAX-SAT

Let $\{C_1, C_2, \ldots, C_m\}$ be a fixed set of disjunctive clauses over a set of k Boolean variables. Each clause C_i is a disjunction of some literals, where a literal is either a Boolean variable or its negation. The online MAX-SAT problem is the following: For each trial t, (i) the player predicts an assignment $\boldsymbol{a}_t \in \{0, 1\}^k$ to the variables, (ii) the adversary returns weights $\boldsymbol{\ell}_t \in [0, 1]^m$ for the clauses, and (iii) the player gets reward defined by the sum of weights $\ell_{t,i}$ over the clauses C_i satisfied by $\boldsymbol{a}_t$. The problem is the *reward version* of a combinatorial online prediction problem

with the concept class $\mathcal{C}$ and *reward* vector space $\mathcal{L}$ as described below. The class $\mathcal{C}$ consists of vectors in $\{0,1\}^n$ for $n = k+m$ such that the first k bit vector $\boldsymbol{a}$ represents the truth assignment and the last m bit vector $\boldsymbol{b}$ represents the truth values of the clauses for the assignment $\boldsymbol{a}$. That is, for each $1 \leq i \leq m$, $b_i = 1$ if and only if C_i is satisfied by $\boldsymbol{a}$. Note that the last m bits $\boldsymbol{b}$ are determined by the first k bits $\boldsymbol{a}$. The reward space $\mathcal{L}$ consists of vectors $0^k\boldsymbol{\ell}$ where the first k bits are 0 and $\boldsymbol{\ell} \in [0,1]^m$ represents the weights. So, the dot product of a concept $\boldsymbol{c} = \boldsymbol{ab}$ and a reward $0^k\boldsymbol{\ell}$ becomes $\boldsymbol{b} \cdot \boldsymbol{\ell}$, which is the reward of the truth assignment $\boldsymbol{a}$ for the weights $\boldsymbol{\ell}$, as required.

The corresponding offline optimization problem is the weighted MAX-SAT problem. The problem has an $3/4$ approximation algorithm using an LP-relaxation and a randomized metarounding [27]. The relaxed space $\mathcal{P} \subseteq [0,1]^n$ is described as the set of feasible solutions $\boldsymbol{x} = \boldsymbol{yz} \in [0,1]^n$ that satisfies the following linear constraints: $\sum_{j \in S_i^+} y_j + \sum_{j \in S_i^-} (1 - y_j) \geq z_i \quad (1 \leq \forall i \leq m)$, where S_i^+ (S_i^-, resp.) denotes the set of Boolean variables occurring nonnegated (negated, resp.) in C_i. The metarounding only computes the first k bit vector $\boldsymbol{a}$ from a relaxed solution $\boldsymbol{y} \in [0,1]^k$ in the following way: Let d be the flip of a fair coin. If $d = 0$, then choose $\boldsymbol{a}$ from $\{0,1\}^n$ uniformly at random, and if $d = 1$, then for each $1 \leq j \leq k$, set $a_j = 1$ with probability y_j and set $a_j = 0$ with probability $1 - y_j$. Note again that the last m bit vector $\boldsymbol{b}$ is determined by $\boldsymbol{a}$. It is shown that the metarounding has approximation factor $3/4$. That is, for any weight vector $\boldsymbol{\ell} \in [0,1]^m$, it holds that $E[\boldsymbol{b} \cdot \boldsymbol{\ell}] \geq (3/4)\boldsymbol{y} \cdot \boldsymbol{\ell}$.

3.4.3 Other Examples and Applications

Other applications include the rank aggregation problem for which metarounding algorithms exist [3, 1]. An online version of the problem was investigated in [84]. Interestingly enough, in this case, $\mathcal{L} \neq [0,1]^n$, for which the pointwise property does not hold.

As far as we know, all rounding methods used in relaxation-based approximation algorithms are metarounding as well. In general, however, there might be concept classes $\mathcal{C}$ for which metarounding algorithms are not known and only α-approximation algorithms are available. Also, one might prefer approximation algorithms with better approximation ratios to metarounding algorithms. In fact, in our experiments, we show a modification of a set covering algorithm, which is not known to metarounding but has better approximation ratio. In the next section, we show a construction of metarounding using α-approximation algorithms.

3.5 Construction of Metarounding Algorithms

In this section, we describe our metarounding algorithms for $\mathcal{L} = [0.1]^n$ using an α-approximation algorithm with the relaxation scheme in Assumption 1 as an oracle. Recall that the approximation algorithm, given $\boldsymbol{\ell} \in \mathcal{L}$, is supposed to output $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{x} \cdot \boldsymbol{\ell}$. The following characterization of metarounding for $\mathcal{L} = [0, 1]^n$ is useful.

Proposition 11. *Suppose that $\mathcal{L} = [0, 1]^n$. A is a metarounding algorithm if and only if given input $\boldsymbol{x} \in \mathcal{P}$, A outputs $\boldsymbol{c} \in \mathcal{C}$ such $E[c_i] \leq \alpha x_i$ for each $i = 1, \ldots, n$.*

Proof. Suppose that A is a metarounding algorithm. Then, for $\boldsymbol{\ell} \in [0, 1]^n$ such that $\ell_i = 1$ for some i and $\ell_j = 0$ for $j \neq i$, it must be that $E[c_i] \leq \alpha x_i$. On the other hand, If $E[c_i] \leq \alpha x_i$, it trivially follows that A is a metarounding algorithm. □

Due to Proposition 11, the problem of constructing a metarounding algorithm is reduced to finding a convex combination $\boldsymbol{\lambda}$ over $\mathcal{C}$ such that $\sum_{\boldsymbol{c} \in \mathcal{C}} \lambda_{\boldsymbol{c}} c_i \leq \alpha x_i$ $(i = 1, \ldots, n)$. That is, by choosing a combinatorial concept $\boldsymbol{c} \in \mathcal{C}$ randomly according to the convex combination $\boldsymbol{\lambda}$, we get that $E[\boldsymbol{c} \cdot \boldsymbol{\ell}] \leq \alpha \boldsymbol{x} \cdot \boldsymbol{\ell}$ for any $\boldsymbol{\ell} \in [0, 1]^n$. Note that the size of $\mathcal{C}$ is exponentially large w.r.t. n in general. Therefore, a naive linear programming formulation over $\mathcal{C}$ to find the convex combination λ would take exponential time.

As noted in the previous section, the first metarounding algorithm was proposed by Carr and Vempala [9]. They formulate a linear program over $\mathcal{C}$ (so the size of the problem could be exponential!). Yet, surprisingly, they showed the problem is solvable by the ellipsoid method (see, e.g, [69]) in polynomial time.

Their theoretical result is quite beautiful, however, might not be practical in the following reasons. First, the ellipsoid method is often much slower in practice, compared to the simplex method or interior point methods. The number of iterations of the ellipsoid method is $O(n^2 \ln \frac{R}{\varepsilon})$, where R is the radius of the initial ellipsoid which contains the feasible region and ε is a precision parameter [2]. Its time complexity per iteration is $O(n^2)$, under the assumption that the running time of the approximation algorithm is constant. The $O(n^4)$ dependence of its time complexity makes the algorithm impractical.

Second, the ellipsoid method requires the knowledge about R which is sometimes not attainable in advance. Setting sufficiently large R would work, but could result in unnecessary

[2] In addition, to achieve this bound, we need to allow the ellipsoid method to violate feasible constraints by amount of ε.

computation. A more detailed treatment of R and ε for rational linear programs is found in, e.g., Schrijver's book [69].

3.5.1 Our formulation

Our formulation is a modification of the original formulation by Carr and Vempala with an additional advantage. The advantage is that our formulation does not require the knowledge of the approximation ratio α of the rounding algorithm. This property is beneficial since, as is often the case, theoretical bounds of the approximation ratio are loose. In other words, our formulation can take advantage of such situations. Our formulation is in problem (3.1). By linear programming duality, the equivalent dual problem turns out to be problem (3.2).

$$\begin{aligned} &\min_{\boldsymbol{\lambda},\beta} \beta && (3.1)\\ \text{s.t.}\quad &\sum_{\boldsymbol{c}\in\mathcal{C}} \lambda_{\boldsymbol{c}} c_i \le \beta x_i \ (i=1,\dots,n),\\ &\sum_{\boldsymbol{c}\in\mathcal{C}} \lambda_{\boldsymbol{c}} \ge 1,\\ &\lambda_{\boldsymbol{c}} \ge 0 \ (\boldsymbol{c}\in\mathcal{C}). \end{aligned}$$

$$\begin{aligned} &\max_{\boldsymbol{\ell},\gamma} \gamma && (3.2)\\ \text{s.t.}\quad &\boldsymbol{c}\cdot\boldsymbol{\ell} \ge \gamma \ (\boldsymbol{c}\in\mathcal{C}),\\ &\boldsymbol{\ell}\cdot\boldsymbol{x} \le 1,\\ &\boldsymbol{\ell} \ge \mathbf{0},\ \gamma \ge 0. \end{aligned}$$

Lemma 2. *Suppose that there exists a rounding algorithm which, given input $\boldsymbol{x}\in\mathcal{P}$, outputs $\boldsymbol{c}\in\mathcal{C}$ s.t. $\boldsymbol{c}\cdot\boldsymbol{\ell} \le \alpha\boldsymbol{x}\cdot\boldsymbol{\ell}$ for any $\boldsymbol{\ell}\in[0,1]^n$. Then, the optimum of problem (3.1) is at most α.*

Proof. We prove this lemma by contradiction. Let $(\boldsymbol{\lambda}^*,\beta^*)$ and $(\boldsymbol{\ell}^*,\gamma^*)$ be optimal solutions of problems (3.1) and (3.2), respectively. Note that, by duality of linear programs, $\beta^* = \gamma^*$. Suppose that $\beta^* = \gamma^* > \alpha$. Then, by using the rounding algorithm with input $\boldsymbol{\ell}^*$, the algorithm outputs some $\boldsymbol{c}\in\mathcal{C}$ such that $\boldsymbol{c}\cdot\boldsymbol{\ell}^* \le \alpha\boldsymbol{x}\cdot\boldsymbol{\ell}^* \le \alpha$. On the other hand, this violates the constraint that $\boldsymbol{c}\cdot\boldsymbol{\ell}^* \ge \gamma^* > \alpha$. So, it contradicts the assumption that $(\boldsymbol{\ell}^*,\gamma^*)$ is an optimal solution. □

3.5.2 Metarounding by Boosting

Now we are ready to describe our algorithm for constructing metarounding. The dual problem (3.2), roughly speaking, can be viewed as the problem of finding a "difficult" loss vector $\boldsymbol{\ell}$ such that $\boldsymbol{c}\cdot\boldsymbol{\ell}$ is large for each $\boldsymbol{c}\in\mathcal{C}$ under some constraints. Here we have access to an α-approximation algorithm with relaxation.Our key observation is that the problem has a similar structure as boosting (e.g., [68]): The problem of boosting is to find a "difficult" distribution over data such for which any base hypotheses (weak hypotheses) have low weighted accuracy,

where the booster has access to weak learner which produce a hypothesis with reasonably small error w.r.t. a given distribution over data.

In fact, we will prove that a boosting-type algorithm works for constructing a metarounding algorithm. Our algorithm, Metarounding by Boosting (MBB), is based on the boosting algorithm SoftBoost [77]. Note that a straightforward application of SoftBoost does not work for our metarounding problem. SoftBoost is applicable for an entropy-maximizing problem of probability distributions with constraints, while our problem deals with non-negative vectors with constraints.

For the time being, we assume that we know an upper bound L on L_ε, defined as $L_\varepsilon = \max\{\|\boldsymbol{\ell}\|_1 \mid \boldsymbol{\ell}$ is an optimal solution of the dual problem (3.2) over $\mathcal{C}' \subseteq \mathcal{C}$ whose solution is at most $\alpha + \varepsilon$. $\}$. L_ε is the constant determined by $\mathcal{P}$, $\mathcal{C}$ and ε. Later, we will explain how to remove this assumption.

The description of MBB is given in Algorithm 6. MBB works in iterations. At each iteration k, MBB solves a modified subproblem of the dual problem (3.2), which is a convex optimization problem. The objective is unnormalized relative entropy from the initial vector $\boldsymbol{\ell}_1$. Note that, by definition of unnormalized relative entropy, any feasible solution satisfies $\boldsymbol{\ell} \geq \mathbf{0}$. So, we can remove the positivity constraint.

Algorithm 6 Metarounding by Boosting (MBB)

Input: $\boldsymbol{x} \in \mathcal{P}$, $L > 0$.

1. Let $\boldsymbol{\ell}_1 = \frac{1}{n}\mathbf{1}$ and let $\mathcal{C}_1 = \emptyset$.
2. For $k = 1, \ldots,$
   (a) Run the approximation algorithm A with input $\boldsymbol{\ell}_k$ and get $\boldsymbol{c}_k \in \mathcal{C}$. Let $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \{\boldsymbol{c}_k\}$ and let $\hat{\gamma}_{k+1} = \max_{j=1,\ldots,k} \boldsymbol{c}_k \cdot \boldsymbol{\ell}_k + \varepsilon$.
   (b) Update $\boldsymbol{\ell}_{k+1}$ as

$$\begin{aligned} \boldsymbol{\ell}_{k+1} = \arg\min_{\boldsymbol{\ell}} \Delta(\boldsymbol{\ell}, \boldsymbol{\ell}_1) \\ \text{sub.to } \boldsymbol{c} \cdot \boldsymbol{\ell} \geq \hat{\gamma}_{t+1}\ (\boldsymbol{c} \in \mathcal{C}_{t+1}),\ \boldsymbol{\ell} \cdot \boldsymbol{x} \leq 1,\ \boldsymbol{\ell} \cdot \mathbf{1} \leq L. \end{aligned} \tag{3.3}$$

   (c) **If** problem (3.3) is infeasible, let $K = k + 1$ and break;
3. Solve problem (3.2) for the reduced set $\mathcal{C}_K$ and output its Lagrange multipliers $\boldsymbol{\lambda}$.

Proposition 12. *For $A, B \geq 0$,*

$$A \ln \frac{A}{B} - A + B \geq \frac{1}{2 \max\{A, B\}} (A - B)^2.$$

Proof. Let $f(x) = x \ln \frac{x}{B} - x + B$. By using Taylor expansion at $x = B$, there exists some B' such that

$$f(x) = f(B) + f'(B)(x - B) + \frac{f''(B')}{2}(x - B)^2 = \frac{1}{2B'}(x - B)^2,$$

where B' satisfies $x < B' < B$ or $B < B' < x$. Since $B' \leq \max\{x, B\}$, we have

$$f(x) \geq \frac{1}{2 \max\{x, B\}} (x - B)^2.$$

By letting $x = A$, we complete the proof. □

Lemma 3. *For each $k = 1, \ldots, K-1$, $\Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_1) - \Delta(\boldsymbol{\ell}_k, \boldsymbol{\ell}_1) \geq \Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_k)$.*

Proof. Let $\mathcal{D}_k$ be the feasible set in problem (3.3) at k-th iteration. Observe that $\hat{\gamma}_k$ is non-decreasing because of $\max$ function. So, we have $\mathcal{D}_{k+1} \subseteq \mathcal{D}_k$. Now, by Generalized Pythagorean Theorem for Bregman divergences (see, e.g., [10]), $\Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_1) \geq \Delta(\boldsymbol{\ell}_k, \boldsymbol{\ell}_1) + \Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_k)$. □

Let $C = \max\{\|\boldsymbol{c}\|_\infty \mid \boldsymbol{c} \in \mathcal{C}\}$. Then the following lemma holds.

Lemma 4. $\Delta(\boldsymbol{\ell}_{t+1}, \boldsymbol{\ell}_t) \geq \frac{\varepsilon^2}{8LC^2}$.

Proof. Let $\bar{c}_{k,i} = c_{k,i}/C$ and $L_k = \sum_{i=1}^n \ell_{k,i}$. Then

$$\Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_k) = \sum_i \ell_{k+1,i} \ln \frac{\ell_{k+1,i}}{\ell_{k,i}} - L_{k+1} + L_k.$$

By decomposing $\ell_{j,i} = \ell_{j,i}\bar{c}_i + \ell_{j,i}(1 - \bar{c}_i)$ for $j = k, k+1$, and and then applying Log-Sum inequality (e.g., [18]), the r. h. s. of the inequality above is lower bounded by

$$p \ln \frac{p}{q} + (P - p) \ln \frac{P - p}{Q - q} - P + Q,$$

where $p = \bar{\boldsymbol{c}} \cdot \boldsymbol{\ell}_{t+1}$ and $q = \bar{\boldsymbol{c}} \cdot \boldsymbol{\ell}_t$, $P = L_{t+1}$, and $Q = L_t$, respectively. Then,

$$\begin{aligned} p \ln \frac{p}{q} + (P - p) \ln \frac{P - p}{Q - q} - P + Q &= p \ln \frac{p/P}{q/Q} \frac{P}{Q} + (P - p) \ln \frac{1 - p/P}{1 - q/Q} \frac{P}{Q} \\ &= P \Delta_2(p/P, q/Q) + P \ln \frac{P}{Q}, \end{aligned}$$

where Δ_2 is called binary relative entropy, i.e., $\Delta_2(a,b) = a\ln\frac{a}{b} + (1-a)\ln\frac{1-a}{1-b}$. By Pinsker's inequality, $\Delta_2(a,b) \geq (a-b)^2/2$ (see,e.g., [77]). Then, by Proposition 12, the first term of the lower bound is further bounded below as

$$P\ln\frac{P}{Q} - P + Q + \frac{P}{2}\left(\frac{p}{P} - \frac{q}{Q}\right)^2 \geq \frac{(P-Q)^2}{2\max\{P,Q\}} + \frac{P}{2}\left(\frac{p}{P} - \frac{q}{Q}\right)^2. \tag{3.4}$$

Then we consider two cases. Suppose that (i) $\frac{P}{Q}q \geq q + \frac{\varepsilon}{2C}$. This assumption is equivalent to the condition that $P \geq Q + \frac{\varepsilon Q}{2qC}$. So, the first term in (3.4) is bounded below by

$$\frac{1}{2\max\{P,Q\}}\left(\frac{\varepsilon Q}{2qC}\right)^2 \geq \frac{\varepsilon^2}{8LC^2},$$

where in the last inequality holds since $q \leq Q$ and $P, Q \leq L$.

Otherwise, it holds that (ii) $\frac{P}{Q}q \leq q + \frac{\varepsilon}{2C}$. This condition implies that $\frac{q}{Q} + \frac{\varepsilon}{2PC} \leq \frac{p}{P}$. So, the second term is at least $\frac{P}{2}\left(\frac{\varepsilon}{2PC}\right)^2 = \frac{\varepsilon^2}{8PC^2} \geq \frac{\varepsilon^2}{8LC^2}$. □

Proposition 13. *For each $k = 1, \ldots, K-1$, $\Delta(\boldsymbol{\ell}_k, \boldsymbol{\ell}_1) \leq O(L\ln Ln)$.*

Proof.

$$\Delta(\boldsymbol{\ell}, \boldsymbol{\ell}_1) = \sum_i^n \ell_i \ln\frac{\ell_i}{L}\frac{L}{1/n} - L + 1 \leq \sum_i^n \ell_i \ln\frac{\ell_i}{L} + L\ln Ln + 1 \leq L\ln Ln + 1.$$

where the inequalities hold since $\sum_i \ell_i \leq L$ and $\ln\frac{\ell_i}{L} \leq 0$. □

Theorem 14. *1. Given a point $\boldsymbol{x} \in \mathcal{P}$, MBB outputs a convex combination $\boldsymbol{\lambda}$ over $\mathcal{C}$ such that*

$$\sum_{\boldsymbol{c}\in\mathcal{C}} \lambda_{\boldsymbol{c}} c_i \leq (\alpha+\varepsilon)x_i \ (i = 1, \ldots, n).$$

2. MBB terminates after

$$K \leq \frac{8L^2C^2\ln Ln}{\varepsilon^2} + 2$$

iterations.

Proof. (i) The algorithm ensures that the problem (3.3) over $\mathcal{C}_K$ is infeasible. So, if we solve the dual problem (3.2) with the restricted set $\mathcal{C}_K$, its solution $(\boldsymbol{\ell}_K^*, \gamma_K^*)$ must satisfy that $\gamma_K^* \leq \hat{\gamma}_K$. Note that, by the property of the approximation algorithm,

$$\hat{\gamma}_K = \max_{k=1,\ldots,K} \boldsymbol{c}_k \cdot \boldsymbol{\ell}_k + \varepsilon \leq \alpha \min_{\boldsymbol{x}'\in\mathcal{P}} \boldsymbol{x}' \cdot \boldsymbol{\ell}_{k^*} + \varepsilon \leq \alpha \boldsymbol{x} \cdot \boldsymbol{\ell}_{k^*} + \varepsilon \leq \alpha + \varepsilon,$$

where $k^* = \arg\max_{k=1,\dots,K} \boldsymbol{c}_k \cdot \boldsymbol{\ell}_k$. Finally, the corresponding primal problem over $\mathcal{C}_K$ has an optimal solution $(\boldsymbol{\lambda}^*_K, \beta^*_K)$ such that $\beta^*_K = \gamma^*_K$ by duality, which completes the proof of the first claim.

(ii)By Lemma 3 for $k = 1, \dots, K-2$ and summing them up, we have

$$\Delta(\boldsymbol{\ell}_{K-1}, \boldsymbol{\ell}_1) - \Delta(\boldsymbol{\ell}_2, \boldsymbol{\ell}_1) \geq \sum_{k=1}^{K-2} \left(\Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_k)\right). \tag{3.5}$$

By Lemma 4, the right hand side of (3.5) is bounded as

$$\sum_{k=1}^{K-2} \left(\Delta(\boldsymbol{\ell}_{k+1}, \boldsymbol{\ell}_k)\right) \geq (K-2)\frac{\varepsilon^2}{8LC^2}. \tag{3.6}$$

Combining Proposition 13 and inequalities (3.5) and (3.6), we have

$$(K-2)\frac{\varepsilon^2}{8LC^2} \leq \Delta(\boldsymbol{\ell}_{K-1}, \boldsymbol{\ell}_1) - \Delta(\boldsymbol{\ell}_2, \boldsymbol{\ell}_1) \leq \Delta(\boldsymbol{\ell}_{K-1}, \boldsymbol{\ell}_1),$$

where the last inequality holds since the unnormalized relative entropy is non-negative. Rearranging this inequality, we complete the proof. □

How to remove the assumption on L So far, we are assuming that $L_\varepsilon \leq L$ for some $L > 1$ which is known. We can remove this assumption by a simple doubling method. Let $L_m = 2^{m-1}$. At each trial $m = 1, \dots$, we run MBB with $L = L_m$. Then, we check if the 1-norm of the dual solution of problem (3.2) over C_K is less than L_m. If the 1-norm is strictly less than L_m, we are done. Otherwise, we let $L = L_{m+1}$ and try this process again. It can be easily verified that the total number of iteration is still $O(L^2_\varepsilon C^2 \ln(L_\varepsilon n)/\varepsilon^2)$.

Time complexity of MBB per iteration is that of convex and linear programs with n variables and $O(\ln n/\varepsilon^2)$ linear constraints, which are solved in polynomial time in n and $1/\varepsilon$. Later, we show that MBB is much faster than the metarounding based on the ellipsoid method in the next section.

3.6 Experiments

We compare performances of two metarounding algorithms, that of the ellipsoid method [9] and MBB on artificial data. Our experiment is performed on a server with four cores of Intel Xeon CPU X5560 2.80GHz and a memory of 198G bytes. We implement programs using Matlab with

Optimization Toolbox. We solve the convex program involved in MBB by sequential quadratic programming.

We generate an artificial data set of set cover instances in the following way. The data set consists of m items and n sets of items. We first add random perturbation to the instances by setting so that for each instance i and each set j, the set j includes the instance i with probability p. Then we fix k "relevant" sets which covers whole the instances. For each instance i, we randomly choose a set among k relevant sets, so that the set covers the instance i. In our experiments, we set $m = 100$, $k = 0.2n$, $p = 0.2$, $n = 10, 50, 100, 200, 500, 1000$, respectively.

We use the simple LP-relaxation based set covering algorithm using deterministic rounding by Hochbaum [34, 80]. The algorithm has a f-approximation guarantee for the LP solution, where $f = \max_{i=1,\dots,m} f_i$ and f_i is the number of sets covering the instance i. The algorithm works as follows. First, the set covering problem is formulated as an integer program. Then, the algorithm solves the LP-relaxation of the problem. Finally, the algorithm rounds the solution $\boldsymbol{p}$ of LP and get the integer solution $\boldsymbol{x}$ by setting $x_i = 1$ if and only if $p_i \geq 1/f$. Note that, one can show that this rounding process is indeed a metarounding. So, we also consider a modification of the algorithm which does not seem to be a metarounding. Our modification is simple. After obtaining the integer solution $\boldsymbol{x}$, we sort each element x_i in the descending order of its associated loss ℓ_i and get $\tilde{\boldsymbol{x}}$. Then, for each $j = 1, \dots, n$, we remove the set j from $\tilde{\boldsymbol{x}}$ as long as the modified vector still represents a set cover.

We generate an internal point $\boldsymbol{x} \in \mathcal{P}$ in the following way. For a random cost vector $\boldsymbol{\ell} \in [0, 1]^n$, we solve the offline set cover problem by using the set covering algorithm above and get a cover $\boldsymbol{c} \in \mathcal{C}$. We repeat this process for 20 iterations and get the average vector of obtained covers.

Given an internal point $\boldsymbol{x} \in \mathcal{P}$, we run metarounding algorithms. For the ellipsoid method, we set $R = n^2$ and $\varepsilon = 0.01$. For MBB, we set $\varepsilon = 0.01$ as well.

Fig. 3.1 shows the computation times (left) and numbers of iterations (right) of metarounding algorithms when we increase the number n of sets. As can be seen in Fig. 3.1, MBB runs about 10^2 or 10^3 times faster than the ellipsoid method. Further, since MBB runs in much fewer iterations, MBB tends to produce much more sparse convex combination of concepts than the ellipsoid method.

Then, we compare the actual approximation ratio β obtained by metarounding algorithms. For the same data sets, we plot the approximation ratios in Fig. 3.2. In addition, we also plot the approximation ratio obtained by MBB with our modified set covering algorithm. MBB achieves

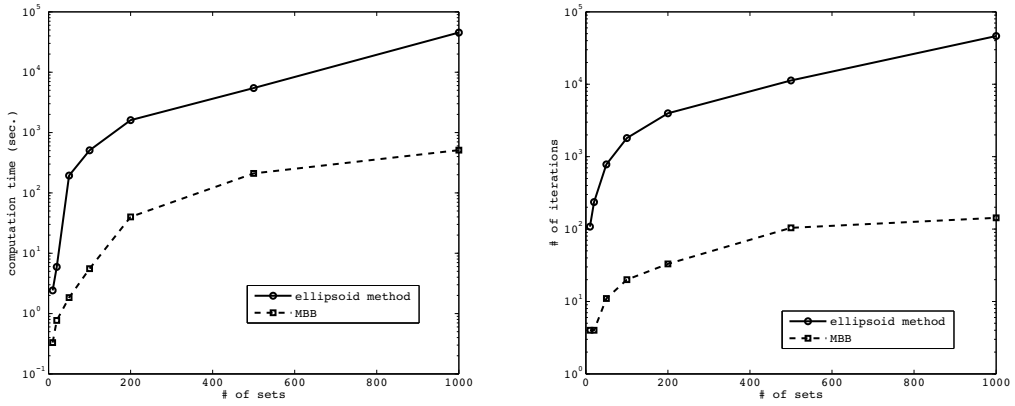

Figure 3.1: Computation times (left, CPU time in seconds) and numbers of iterations (right) of metarounding algorithms.

better approximation ratios than the ellipsoid method. By using the modified algorithm, MBB further gains better ratios than MBB with Hochbaum's original algorithm. Therefore MBB can take advantage of the situation where actual the approximation ratio of the algorithm is better than theoretically guaranteed.

3.7 Conclusion

In this thesis, we proposed algorithms for online combinatorial prediction using metarounding algorithms. Also, we showed an efficient construction method of metarounding algorithms using a relaxation-based approximation algorithm as an oracle. Our algorithm is adaptive in the sense that it does not require the explicit knowledge on the approximation ratio of the approximation algorithm. Also, computation time of our algorithms at each trial do not depend on T, unlike previous methods.

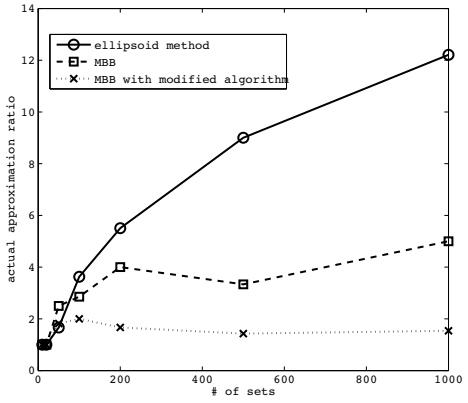

Figure 3.2: Actual approximation ratios obtained by metarounding algorithms, the ellipsoid method and MBB with Hochbaum's original set covering algorithm and MBB with our modified set covering algorithm.

Chapter 4

Online Combinatorial Optimization with Multiple Projections and Its Application to Scheduling Problem

4.1 Introduction

We study online linear optimization problems over combinatorial concept classes $\mathcal{C} \subseteq \mathbb{R}^n$ that are defined in some combinatorial ways. Examples of such classes are s-t paths in a given graph, spanning trees of a given graph, permutations over a given set, truth assignments for a given CNF formula, set covers of a given subset family, and so on. Typically, those concept classes are finite but contain exponentially many concepts. The problem for a concept class $\mathcal{C}$ is described as follows: For each trial $t = 1, ..., T$, (i) the player guesses $\boldsymbol{c}_t \in \mathcal{C}$, (ii) the adversary returns a loss vector $\boldsymbol{\ell}_t \in [0, 1]^n$, and (iii) the player incurs loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$. A typical goal of the player is to minimize the regret: $\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t$. The goal of the player is to minimize the α-regret for some small $\alpha \geq 1$:

$$\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t.$$

The α-regret measures difference between the cumulative loss of the player and α times that of the best fixed concept in hindsight. In particular, when $\alpha = 1$, 1-regret is sometimes called regret. The second term can be viewed as the cumulative loss of the fixed concept given by an α-approximation (offline) algorithm which takes $\sum_{t=1}^{T} \boldsymbol{\ell}_t$ as input and returns $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c} \cdot (\sum_{t=1}^{T} \boldsymbol{\ell}_t) \leq \alpha \min_{\boldsymbol{c}^* \in \mathcal{C}} \boldsymbol{c}^* (\sum_{t=1}^{T} \boldsymbol{\ell}_t)$. In other words, the goal of the player is to predict as

well as the best α-approximation offline algorithm in hindsight.

One of main approaches in the combinatorial online prediction is to relax the problem to the online convex optimization, where the decision set is a continuous convex set $\mathrm{Relax}(\mathcal{C})$ (typically, the convex hull of $\mathcal{C}$) rather than a discrete set, and then to round the "fractional" real prediction to a discrete combinatorial concept. There are many results following this approach, e.g., for k-sets [78], permutations [32, 83], s-t paths [40], spanning trees [74] and so on. This approach consists of three components: (i) an online prediction algorithm for real vectors (ii) projection onto the relaxed domain, and (iii) rounding method from real vectors to combinatorial concepts. For (i), we can employ many existing algorithms for the online convex optimization such as the OGD [86] and the Hedge [19] and etc. For (ii) and (iii), we need to design specific algorithms for each class of combinatorial concepts [1] In particular, the projection step onto the relaxed set $\mathrm{Relax}(\mathcal{C})$ is computationally cumbersome in general, since the constraints are sometimes exponentially many. Further, if we consider a set of combinatorial concepts $\mathcal{C}$ with constraints, $\mathcal{C} \cap \mathcal{A}$, where $\mathcal{A}$ is a set of constraints, the underlying projection onto $\mathrm{Relax}(\mathcal{C}) \cap \mathcal{A}$ become harder to compute.

Our technical contributions have two folds. First, we propose an alternative framework for the relaxation-rounding approach for the combinatorial online prediction. Our framework allows multiple projections onto a series of sets $\mathrm{Relax}(\mathcal{C}) \cap \mathcal{A}_1 \supseteq \mathrm{Relax}(\mathcal{C}) \cap \mathcal{A}_2, , ..., \supseteq \mathrm{Relax}(\mathcal{C}) \cap \mathcal{A}$. Whereas the last projection is not necessarily the same as the direct projection, we show essentially the same regret bounds with those with a single projection. So, our framework is useful when a series of sequential projections are computationally advantageous to a single direct projection.

Second, we exhibit a case where the multiple projections are indeed useful in a problem of online scheduling of n jobs with a single machine under precedence constraints. In the problem, we devise double projections to the sets of interest, where there is no efficient algorithm known for the direct projection. More precisely, we propose an algorithm for the online job scheduling with α-regret $O(n^2\sqrt{T})$ for $\alpha = 2 - 2/(n+1)$. The time complexity of the algorithm per trial is $O(n^4)$. Further, we show that a lower bound of the 1-regret is $\Omega(n^2\sqrt{T})$.

In addition, we prove that there is no polynomial time algorithm with α-regret $poly(n, m)\sqrt{T}$ with $\alpha < 2 - 2/(n+1)$ unless there exists a randomized approximation algorithm with approximation $\alpha < 2 - 2/(n+1)$ for the corresponding offline problem. Thus far, the state-of-the-art

[1] As an exception, Suehiro et al. proposed generic projection and rounding algorithms for several classes of combinatorial concepts parameterized by submodular functions [74].

approximation algorithms have an approximation ratio $2 - 2/(n+1)$, and it is a longstanding open problem to find an approximation algorithm with a better ratio [82]. It has been determined that there is no polynomial-time $(1+\varepsilon)$-approximation algorithm (PTAS) for any constant $\varepsilon > 0$ under some standard assumption of the complexity theory [5]. Therefore, the regret bound is optimal among any polynomial algorithms unless there exists a better approximation algorithm for the offline problem.

4.1.1 Online job scheduling with precedence constraints

We consider an online job scheduling problem on a single machine with precedence constraints under uncertainty. Each day $t = 1, \ldots, T$, we determine a total order of n fixed jobs satisfying some prefixed precedence constraints. Then, after processing all n jobs according to the schedule, the processing time of each job is revealed. The goal is to minimize *the sum of the completion times* over all jobs and all T days under fixed precedence constraints, where the completion time of job i at day t is the sum of processing times of all jobs prior to i and the processing time of job i.

Now, let us formulate the problem in a formal way. A permutation σ is a bijection from $[n]$ to $[n]$. Another representation of a permutation $\boldsymbol{\sigma}$ over the set $[n]$ is a vector in $[n]^n$, defined as $\boldsymbol{\sigma} = (\sigma(1), \ldots, \sigma(n))$, which corresponds to σ. For example, $(3, 2, 1, 4)$ is a representation of a permutation for $n = 4$. The vector representation of permutations is convenient, since the sum of the completion times of the jobs according to some permutation $\boldsymbol{\sigma}$ is expressed as the inner product $\boldsymbol{\sigma} \cdot \boldsymbol{\ell}$, where $\boldsymbol{\ell}$ is the vector consisting of the processing times of the jobs. For example, there are 4 jobs to be processed according to the order $4, 1, 2, 3$. Each processing time is given as $\boldsymbol{\ell} = (\ell_1, \ell_2, \ell_3, \ell_4)$. The completion times of jobs $i = 4, 1, 2, 3$ are ℓ_4, $\ell_4 + \ell_1$, $\ell_4 + \ell_1 + \ell_2$, and $\ell_4 + \ell_1 + \ell_2 + \ell_3$, respectively, and the sum of completion times is $4\ell_4 + 3\ell_1 + 2\ell_2 + \ell_3$. Note that the completion time exactly corresponds to $\boldsymbol{\sigma} \cdot \boldsymbol{\ell}$, the inner product of $\boldsymbol{\ell}$ and the permutation vector $\boldsymbol{\sigma} = (3, 2, 1, 4)$. Here, component σ_i of the permutation $\boldsymbol{\sigma}$ represents the priority of job i.

Let S_n be the set of all permutations over $[n]$, i.e., $S_n = \{\boldsymbol{\sigma} \in [n]^n \mid \boldsymbol{\sigma}$ is a permutation over $[n]\}$. In particular, the convex hull of all permutations is called the permutahedron, denoted as P_n. Figure 4.1 illustrates the fourth-order permutahedron P_4.

The set $\mathcal{A}$ of precedence constraints is given as $\mathcal{A} = \{(i_k, j_k) \in [n] \times [n] \mid i_k \neq j_k, k = 1, \ldots, m\}$, meaning that object i_k is preferred to object j_k. The set $\mathcal{A}$ induces the set defined by

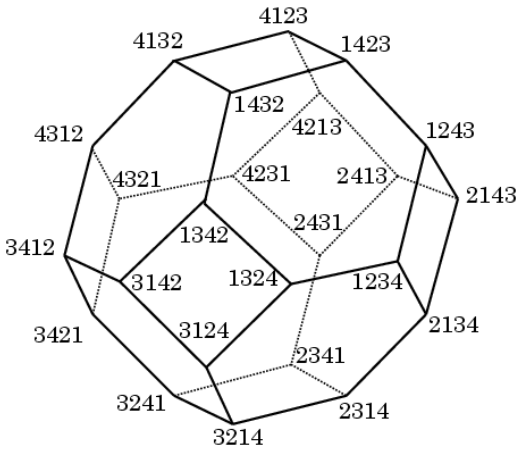

Figure 4.1: the fourth-order permutahedron P_4.

the linear constraints $\mathrm{Precons}(\mathcal{A}) = \{\boldsymbol{p} \in \mathbb{R}^n_+ \mid p_i \geq p_j \text{ for } (i,j) \in \mathcal{A}\}$. We further assume that there exists a linear ordering consistent with $\mathcal{A}$. In other words, we assume there to exist a permutation $\boldsymbol{\sigma} \in S_n \cap \mathrm{Precons}(\mathcal{A})$.

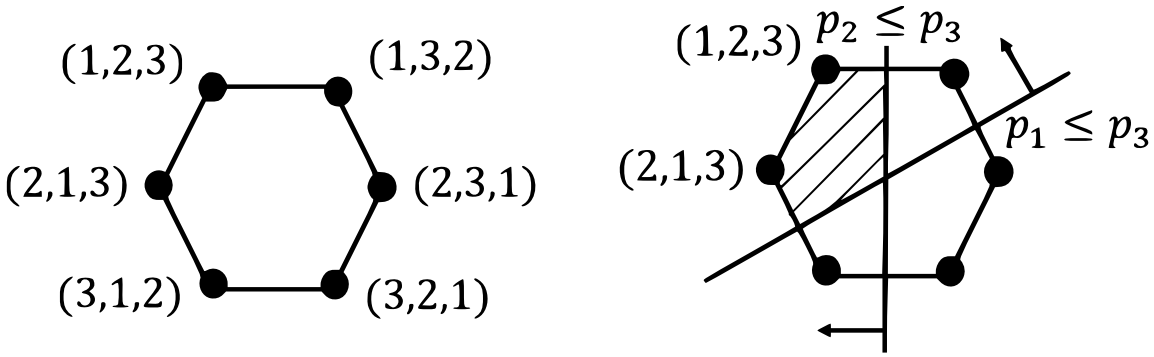

Figure 4.2: (i) P_3; (ii) $S_3 \cap \mathrm{Precons}(\mathcal{A})$

The online job scheduling problem can be formulated as the following online linear optimization problem over $S_n \cap \mathrm{Precons}(\mathcal{A})$. For each trial $t = 1, \ldots, T$, (i) the player predicts a permutation $\boldsymbol{\sigma}_t \in S_n \cap \mathrm{Precons}(\mathcal{A})$, (ii) the adversary returns a loss vector $\boldsymbol{\ell}_t \in [0,1]^n$, and (iii) the player incurs loss $\boldsymbol{\sigma}_t \cdot \boldsymbol{\ell}_t$. The goal of the player is to minimize the α-regret for some small $\alpha \geq 1$:

$$\alpha\text{-Regret} = \sum_{t=1}^{T} \boldsymbol{\sigma}_t \cdot \boldsymbol{\ell}_t - \alpha \min_{\boldsymbol{\sigma} \in S_n \cap \mathrm{Precons}(\mathcal{A})} \sum_{t=1}^{T} \boldsymbol{\sigma} \cdot \boldsymbol{\ell}_t.$$

We will use the notion of the permutahedron. The permutahedron P_n is the convex hull of the set of permutations S_n. It is known that P_n can be represented as the set of points $\boldsymbol{p} \in \mathbb{R}^n_+$

satisfying $\sum_{i \in S} p_i \leq \sum_{i=1}^{|S|}(n+1-i)$ for any $S \subset [n]$, and $\sum_{i=1}^{n} p_i = n(n+1)/2$. For further discussion of the permutahedron, see, e.g., [85, 21].

4.1.2 Related Work

Other approaches in the combinatorial online prediciton

A naive approach for combinatorial online prediction problems is to apply the Hedge algorithm [19]. Given a set of experts (i.e., prediction strategies or algorithms), Hedge algorithm is shown to predict almost as well as the best expert in hindsight. Therefore, with each combinatorial concept $\boldsymbol{c} \in \mathcal{C}$ as an expert, Hedge algorithm achieves low regret. This approach, however, is inefficient in general since there are exponentially many concepts in the class $\mathcal{C}$ and thus the Hedge algorithm takes exponential time at each trial. Yet, for some classes of concepts, polynomial time simulations of the Hedge are possible (see, e.g., [76] for s-t paths and [11] for permutations, spanning trees and so on).

The other main approach for combinatorial online prediction is to convert offline approximation algorithms for combinatorial optimization to online prediction algorithms. More precisely, we assume an offline α-approximation algorithm A for the linear optimization problem over $\mathcal{C}$. There are three main previous researches on the player's strategy for combinatorial online prediction problems where approximation algorithms are available. First, Kalai and Vempala proposed Follow the perturbed leader (FPL [38]). The 1-regret bound of FPL is $O(\sqrt{T})$ and its running time per trial is $O(n)$ when a call of the offline algorithm is done in a unit time. FPL also works with α-approximation algorithms, however, its α-regret bound becomes $O(\alpha^T\sqrt{T})$ in general. Second, Kakade et al. proposed a different strategy using α-approximation algorithms, which achieves $O(\alpha\sqrt{T})$ α-regret bound [37]. The running time of the algorithm is $O(poly(n)T)$. Unfortunately, the time complexity at each trial depends on T, which is not desirable in practice. Finally, Fujita et al. [24] show that if the approximation algorithm is based on relaxation scheme, we can construct an efficient online algorithm with low $(\alpha+\epsilon)$-regret for any constant $\epsilon > 0$. However, the running time per trial is $poly(n, \frac{1}{\epsilon})$, which is independent of T but depends on $\frac{1}{\epsilon}$.

Offline algorithms for job scheduling with a single machine under precedence constraints

In offline version of the problem, given the processing time of n jobs and the precedence constraints $\mathcal{A}$, the typical goal is to find a schedule that minimizes the sum of the (weighted)

completion times of the n jobs, subject to the constraints $\mathcal{A}$. This problem is categorized as $1|prec|\sum_j C_j$ in the literature[2]. It is known that this problem is NP-hard [43, 44]. For further developments, see, e.g., [17, 4]. Several 2-approximation algorithms have been proposed for the offline setting [70, 29, 14, 52, 12] and a stochastic setting [58, 71].

4.2 Online Combinatorial Prediction with Multiple Projections

In this section, we propose a generic algorithm with multiple projections and prove its α-regret bound.

Throughout the section, we put the following assumptions.

- There exist a number K of closed convex sets $\mathrm{Relax}_k(\mathcal{C})$ $(k = 1, \ldots, K)$ such that (i) $\mathcal{X} \stackrel{\text{def}}{=} \mathrm{Relax}_0 \supset \mathrm{Relax}_1(\mathcal{C}) \supset \mathrm{Relax}_2(\mathcal{C}) \supset \cdots \supset \mathrm{Relax}_K(\mathcal{C}) \supset \mathcal{C}$, and (ii) for any $k = 1, \ldots, K$, the Bregman Projection from $\mathrm{Relax}_{k-1}(\mathcal{C})$ to $\mathrm{Relax}_k(\mathcal{C})$ is feasible. More precisely, for any $\boldsymbol{x}' \in \mathrm{Relax}_{k-1}(\mathcal{C})$, $\min_{\boldsymbol{x} \in \mathrm{Relax}_k(\mathcal{C})} \Delta_\Phi(\boldsymbol{x}, \boldsymbol{x}')$ can be computed in polynomial time in n.

- $\mathcal{C}$ has an efficient α-metarounding algorithm from $\mathrm{Relax}_K(\mathcal{C})$ for some $\alpha \geq 1$. More precisely, there exists a polynomial time algorithm that, when given any $\boldsymbol{x} \in \mathrm{Relax}_K(\mathcal{C})$, outputs $\boldsymbol{c} \in \mathcal{C}$ such that for any loss vector $\boldsymbol{\ell} \in \mathbb{R}^n_+$, $\boldsymbol{c} \cdot \boldsymbol{\ell} \leq \alpha \boldsymbol{x} \cdot \boldsymbol{\ell}$.

- The diameter of $\mathrm{Relax}_K(\mathcal{C})$ is upper bounded by D with respect to some norm $\|\cdot\|$ and all loss vectors $\boldsymbol{\ell}_t$ $(t = 1, \ldots, T)$ returned by the adversary have bounded norm, i.e., $\|\boldsymbol{\ell}_t\|_* \leq G_*$ for some G_* where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$.

Note that the notion of metarounding differs from the rounding in that the algorithm outputs a rounded vector (concept) whose loss is small for any loss vector without seeing the loss vector[3].

Our algorithm is shown as Algorithm 7. The algorithm consists of three components, an online prediction algorithm, multiple projections, and a "metarounding" algorithm.

Our online prediction algorithm is based on the standard algorithm the Online Mirror Descent (see, e.g., [10, 30], OMD for short) for the online convex optimization. The OMD is a

[2]The weighted version is known as $1|prec|\sum_j w_j C_j$.

[3]The notion was first formalized by Carr and Vempala [9] for optimization of a generalized congestion problem and later it was shown to be useful in online combinatorial prediction [24].

generalization of the Online Gradient Descent (OGD[86]). Given a uniformly separable strictly convex function Φ, at each trial t, the OMD updates a vector $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1/2}$ by mapping $\boldsymbol{x}_t$ using $\nabla\Phi : \mathbb{R}^n \to \mathbb{R}^n$ (note that $\nabla\Phi$ is a bijection because of the strict convexity of Φ) to some "parameter space", performing an additive update $\boldsymbol{y}_t = \Phi(\boldsymbol{x}_t) - \eta\boldsymbol{\ell}_t$ and mapping back $\boldsymbol{y}_t$ with $\nabla\Phi^{-1}$.

Typical versions of the OMD uses a single projection onto the space of interest. On the other hand, our algorithm uses multiple successive projections from $\mathcal{X}$ to $\mathrm{Relax}(\mathcal{C})$ defined in Algorithm 8.

The meterounding algorithm, at each trial t, chooses $\boldsymbol{c}_t \in \mathcal{C}$ by rounding $\boldsymbol{x}_t$, where, roughly speaking, $\boldsymbol{c}_t$ is close to $\boldsymbol{x}_t$. More precisely, we say that an algorithm A is an α-metarounding w.r.t. $\mathcal{C}$ and $\mathrm{Relax}(\mathcal{C}) \supseteq \mathcal{C}$ if, for any loss vector $\boldsymbol{\ell} \in [0,1]^n$ and any $\boldsymbol{x} \in \mathrm{Relax}(\mathcal{C})$, A, given $\boldsymbol{x}$ as an input, outputs $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c}\cdot\boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{x}^* \in \mathrm{Relax}(\mathcal{C})} \boldsymbol{x}^*\cdot\boldsymbol{\ell}$. The notion of metarounding differs from rounding in that the algorithm outputs a rounded vector which is effective for any loss vector without using the information of the loss vector[4].

After the loss vector $\boldsymbol{\ell}_t$ is given, our algorithm updates the vector $\boldsymbol{x}_t$ in an additive way and multiple projects it onto the sets $\mathrm{Relax}_k(\mathcal{C})$ for $k = 1, \cdots, K$.

Algorithm 7 PUMMA

Input: parameter $\eta > 0$, a separable strictly convex function Φ.

1. Let $\boldsymbol{x}_0$ be such that $\nabla\Phi(\boldsymbol{x}_0) = \boldsymbol{0}$ and $\boldsymbol{x}_1 = \arg\min_{\boldsymbol{x}\in\mathrm{Relax}_K(\mathcal{C})} \Delta_\Phi(\boldsymbol{x}, \boldsymbol{x}_0)$.
2. For $t = 1, \ldots, T$
   (a) (Metarounding) get $\boldsymbol{c}_t \in \mathcal{C}$ such that $\boldsymbol{c}_t \leq \alpha\boldsymbol{x}_t$.
   (b) Incur a loss $\boldsymbol{c}_t \cdot \boldsymbol{\ell}_t$.
   (c) Update $\boldsymbol{x}_{t+1/2}$ according to the rule:

$$\begin{aligned} \boldsymbol{y}_t &= \nabla\Phi(\boldsymbol{x}_t) - \eta\boldsymbol{\ell}_t. \\ \boldsymbol{x}_{t+1/2} &= \nabla\Phi^{-1}(\boldsymbol{y}_t). \end{aligned}$$

   (d) (Multiple Projection) $\boldsymbol{x}_{t+1}$ =MultipleProjection($\boldsymbol{x}_{t+1/2}$).

First, we will prove an α-regret bound of the proposed algorithm. We begin our analysis of the projection algorithms with the following lemma.

[4]The notion was first formalized by Carr and Vempala [9] for optimization of a generalized congestion problem and later it was shown to be useful in online combinatorial prediction [24].

Algorithm 8 MultipleProjection

Input: $\boldsymbol{x} \in \mathrm{Relax}(\mathcal{C})$

1. Let $\boldsymbol{x}^{(0)} = \boldsymbol{x}$ and $\mathrm{Relax}_0(\mathcal{C}) = \mathcal{X}$.
2. For $k = 1, \ldots, K$:
   (a) Let $\boldsymbol{x}^{(k)}$ be the Bregman projection of $\boldsymbol{x}^{(k-1)} \in \mathrm{Relax}_{k-1}(\mathcal{C})$ w.r.t. Φ onto $\mathrm{Relax}_k(\mathcal{C})$, .i.e.,
   $$\boldsymbol{x}^{(k)} = \arg \min_{\boldsymbol{x} \in \mathrm{Relax}_k(\mathcal{C})} \Delta_\Phi(\boldsymbol{x}, \boldsymbol{x}^{(k-1)}).$$
3. Output $\boldsymbol{x}^{(K)}$.

Lemma 5. *For any $\boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C})$ and for any $t \geq 1$,*

$$\Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}_{t+1/2}) \geq \Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}_{t+1})$$

Proof. For simplicity of notations, let $\boldsymbol{x}_{t+1/2} = \boldsymbol{x}^{(0)}$ and $\boldsymbol{x}_{t+1} = \boldsymbol{x}^{(K)}$. For $k = 1, \ldots, K$, let $\boldsymbol{x}^{(k)}$ be the Bregman projection of $\boldsymbol{x}^{(k-1)} \in \mathrm{Relax}_{k-1}(\mathcal{C})$ w.r.t. Φ onto $\mathrm{Relax}_k(\mathcal{C})$. By using Theorem 1, for $k = 1, \cdots, K$,

$$\begin{aligned}\Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}^{(k-1)}) &\geq \Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}^{(k)}) + \Delta_\Phi(\boldsymbol{x}^{(k)}, \boldsymbol{x}^{(k-1)}) \\ &\geq \Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}^{(k)}),\end{aligned}$$

where the last inequality follows since the Bregman divergence is nonnegative. By combining the inequality above for $k = 1, \ldots, K$, we complete the proof. □

The following lemma shows that the OMD using multiple projection can achieve $O(\sqrt{T})$ bounds on the regret.

Lemma 6 (Cf. Hazan [30])**.** *Suppose that for all $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{X}$.$\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) \geq \|\boldsymbol{p} - \boldsymbol{q}\|^2/2$ for some norm $\|\cdot\|$. Let $\|\boldsymbol{\ell}_t\|_* \leq G_*$ for $t = 1, \ldots, T$, where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$, and $\forall \boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C}), \Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}_1) \leq D^2/2$. For any $T \geq 1$ and $\eta = D/(G_*\sqrt{T})$,*

$$\sum_{t=1}^{T} \boldsymbol{x}_t \cdot \boldsymbol{\ell}_t \leq \min_{\boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C})} \sum_{t=1}^{T} \boldsymbol{x}^* \cdot \boldsymbol{\ell}_t + DG_*\sqrt{T}.$$

Proof. The proof partly follows [30]. The following property of Bregman divergences follows

easily from the definition: for any vectors $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r} \in \mathcal{X}$,

$$\begin{aligned}&(\boldsymbol{p}-\boldsymbol{q})\cdot(\nabla\Phi(\boldsymbol{r})-\nabla\Phi(\boldsymbol{q}))\\&=\Delta_\Phi(\boldsymbol{p},\boldsymbol{q})-\Delta_\Phi(\boldsymbol{p},\boldsymbol{r})+\Delta_\Phi(\boldsymbol{q},\boldsymbol{r}).\end{aligned}$$

Then, for all $\boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C}) \subset \mathcal{X}$, by letting $\boldsymbol{p}=\boldsymbol{x}^*$, $\boldsymbol{q}=\boldsymbol{x}_t$, and $\boldsymbol{r}=\boldsymbol{x}_{t+1/2}$,

$$\begin{aligned}&(\boldsymbol{x}_t-\boldsymbol{x}^*)\cdot\boldsymbol{\ell}_t\\&=\frac{1}{\eta}(\boldsymbol{x}^*-\boldsymbol{x}_t)\cdot(\nabla\Phi(\boldsymbol{x}_{t+1/2})-\nabla\Phi(\boldsymbol{x}_t))\\&=\frac{1}{\eta}(\Delta_\Phi(\boldsymbol{x}^*,\boldsymbol{x}_t)-\Delta_\Phi(\boldsymbol{x}^*,\boldsymbol{x}_{t+1/2})+\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2}))\\&\le\frac{1}{\eta}(\Delta_\Phi(\boldsymbol{x}^*,\boldsymbol{x}_t)-\Delta_\Phi(\boldsymbol{x}^*,\boldsymbol{x}_{t+1})+\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2})),\end{aligned}$$

where the last inequality follows Lemma 5. Summing over all iterations,

$$\begin{aligned}&\sum_{t=1}^T\boldsymbol{x}_t\cdot\boldsymbol{\ell}_t-\min_{\boldsymbol{x}^*\in\mathrm{Relax}_K(C)}\sum_{t=1}^T\boldsymbol{x}^*\cdot\boldsymbol{\ell}_t\\&\le\frac{1}{\eta}(\Delta_\Phi(\boldsymbol{x},\boldsymbol{x}_1)-\Delta_\Phi(\boldsymbol{x},\boldsymbol{x}_{T+1}))+\sum_{t=1}^T\frac{1}{\eta}\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2})\\&\le\frac{1}{\eta}D^2+\sum_{t=1}^T\frac{1}{\eta}\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2}),\end{aligned}\tag{4.1}$$

Now we proceed to bound $\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2})$. By definition of the Bregman divergence,

$$\begin{aligned}&\Delta_\Phi(\boldsymbol{x}_t,\boldsymbol{x}_{t+1/2})+\Delta_\Phi(\boldsymbol{x}_{t+1/2},\boldsymbol{x}_t)\\&=(\nabla\Phi(\boldsymbol{x}_t)-\nabla\Phi(\boldsymbol{x}_{t+1/2})\cdot(\boldsymbol{x}_t-\boldsymbol{x}_{t+1/2})\\&=\eta\boldsymbol{\ell}_t\cdot(\boldsymbol{x}_t-\boldsymbol{x}_{t+1/2})\\&\le\eta\|\boldsymbol{\ell}_t\|^*\|\boldsymbol{x}_t-\boldsymbol{x}_{t+1/2}\|\ \text{(by Hölder's inequality)}\\&\le\frac{1}{2}\left(\eta\|\boldsymbol{\ell}_t\|_*^2+\|\boldsymbol{x}_t-\boldsymbol{x}_{t+1/2}\|^2\right)\\&\le\frac{1}{2}(\eta^2G_*^2+\|\boldsymbol{x}_t-\boldsymbol{x}_{t+1/2}\|^2),\end{aligned}$$

where the last inequality holds by the assumption on $\|\boldsymbol{\ell}_t\|_*$. Thus, by the assumption that

$\Delta_\Phi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1/2}) \geq \|\boldsymbol{x}_t - \boldsymbol{x}_{t+1/2}\|^2/2$, we have

$$\begin{aligned}
&\Delta_\Phi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1/2}) \\
&\leq \frac{1}{2}(\eta^2 G_*^2 + \|\boldsymbol{x}_t - \boldsymbol{x}_{t+1/2}\|^2) - \Delta_\Phi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1/2}) \\
&\leq \frac{1}{2}\eta^2 G_*^2.
\end{aligned}$$

Plugging back into Equation (1), and by the non-negativity of the Bregman divergence, we get

$$\begin{aligned}
&\sum_{t=1}^{T} \boldsymbol{x}_t \cdot \boldsymbol{\ell}_t - \min_{\boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C})} \sum_{t=1}^{T} \boldsymbol{x}^* \cdot \boldsymbol{\ell}_t \\
&\leq \frac{1}{2}(\frac{1}{\eta} D^2 + \eta T G_*^2) \\
&\leq D G_* \sqrt{T}
\end{aligned}$$

by taking $\eta = \frac{D}{G_*\sqrt{T}}$. □

We are now ready to prove the main result.

Theorem 15. *There exists an online combinatorial optimization algorithm over $\mathcal{C}$ such that its α-regret is $O(\alpha D G^* \sqrt{T})$.*

Proof. By definition of the meterounding and Lemma 6, the cumulative loss of Algorithm 7 is

$$\begin{aligned}
\sum_{t=1}^{T} \boldsymbol{c}_t \cdot \boldsymbol{\ell}_t &\leq \alpha \sum_{t=1}^{T} \boldsymbol{x}_t \cdot \boldsymbol{\ell}_t \\
&\leq \alpha \min_{\boldsymbol{x}^* \in \mathrm{Relax}_K(\mathcal{C})} \sum_{t=1}^{T} \boldsymbol{x}^* \cdot \boldsymbol{\ell}_t + D G_* \sqrt{T} \\
&\leq \alpha \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{c} \cdot \boldsymbol{\ell}_t + \alpha D G_* \sqrt{T},
\end{aligned}$$

where the last inequality holds since $\mathcal{C} \subseteq \mathrm{Relax}_K(\mathcal{C})$. □

4.3 Application: Online job scheduling with precedence constraints

In this section, we consider online job scheduling with precedence constraints. In this case, the set $\mathcal{C}$ of combinatorial concepts corresponds to $S_n \cap \mathrm{Precons}(\mathcal{A})$, the set of permutations

satisfying some precedence constraints. Note that, for the convex hull P_n of the set of permutations S_n, the set $P_n \cap \mathrm{Precons}(\mathcal{A})$ is a closed convex set described with linear constraints and satisifies $P_n \cap \mathrm{Precons}(\mathcal{A}) \supset \mathcal{C}$. However, the Bregman projection onto $P_n \cap \mathrm{Precons}(\mathcal{A})$ is not known to be tractable, and it contains exponentially many linear constraints. Thus, we take our approach with multiple projections given in the previous section. Instead of performing the projection directly, we use successive projections onto $\mathrm{Precons}(\mathcal{A})$ and $P_n \cap \mathrm{Precons}(\mathcal{A})$. That is, we set $\mathrm{Relax}_1(\mathcal{C}) = \mathrm{Precons}(\mathcal{A})$, and $\mathrm{Relax}_2(\mathcal{C}) = P_n \cap \mathrm{Precons}(\mathcal{A})$, and apply an online prediction algorithm with the metarounding and the double projections. Below, we will show that these successive projections are the key to an efficient implementation of our algorithm.

4.3.1 Implementations of the OMD

For the online scheduling problem, we employ the OMD with the squared 2-norm $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$. This version is known to be the OGD ([86]). Then the update at each trial can be simplified as $\boldsymbol{x}_{t+1/2} = \boldsymbol{x}_t - \eta\boldsymbol{\ell}_t$. The Bregman divergence corresponds to $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{q}\|_2^2$. By setting $\boldsymbol{x}_0 = \mathbf{0}$ and $\boldsymbol{x}_1 = Metarounding(\boldsymbol{x}_0)$, for any $\boldsymbol{x}^* \in \mathrm{Relax}(\mathcal{C})$, $\Delta_\Phi(\boldsymbol{x}^*, \boldsymbol{x}_1) = O(n^{\frac{3}{2}})$, where we can set $D^2 = O(n^2)$. Also, since the dual norm of the 2-norm is the 2-norm itsef, $\|\boldsymbol{\ell}_t\|_* = \|\boldsymbol{\ell}_t\|_2 = O(\sqrt{n})$. So we can set $G_* = O(\sqrt{n})$. Therefore, by applying Lemma 6, the regret of our algorithm is $O(n^2\sqrt{T})$.

4.3.2 Efficient Implementations of Multiple Projections

In this section, we propose efficient algorithms for multiple projections onto $\mathrm{Precons}(\mathcal{A})$ and $P_n \cap \mathrm{Precons}(\mathcal{A})$. We then show an implementation of the procedure Metarounding.

Projection onto the Set $\mathrm{Precons}(\mathcal{A})$ of the Precedence Constraints

The problem of projection onto $\mathrm{Precons}(\mathcal{A})$ is described as follows:

$$\begin{aligned} &\min_{\boldsymbol{p}\in\mathbb{R}^n} \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) \\ &\text{sub.to: } p_i \geq p_j, \quad \text{for } (i,j) \in \mathcal{A}. \end{aligned}$$

This problem is known as the isotonic regression problem [53, 72, 50, 49]. Previously known algorithms for the isotonic regression run in $O(mn^2 \log n)$ or $O(n^4)$ time (see [49] for details), where $m = |\mathcal{A}|$.

Projection of a point in $\mathrm{Precons}(\mathcal{A})$ **onto** $P_n \cap \mathrm{Precons}(\mathcal{A})$

In this subsection, we show an efficient algorithm for computing the projection a point in $\mathrm{Precons}(\mathcal{A})$ onto the intersection of the permutahedron P_n and the set $\mathrm{Precons}(\mathcal{A})$ of the precedence constraints. In fact, we will show that the problem can be reduced to projection onto P_n only, and thus we can use the algorithm of Lim et al. [46] for finding the projection onto P_n in time $O(n \log n)$ for the uniformly separable Bregman divergences. Also, a simpler $O(n^2)$ projections algorithms for the Euclidean distance and the relative entropy are given in [74] and [83], respectively.

Formally, the problem is stated as follows:

$$\begin{aligned} &\min_{\boldsymbol{p}\in\mathbb{R}^n} \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) \\ \text{sub. to: } &\sum_{j\in S} p_j \le \sum_{j=1}^{|S|} (n+1-j), \text{ for any } S \subset [n], \\ &\sum_{j=1}^{n} p_j = \frac{n(n+1)}{2}, \\ &p_i \ge p_j, \quad \text{for } (i,j) \in \mathcal{A}. \end{aligned}$$

Without loss of generality, we may assume that elements in $\boldsymbol{q}$ are sorted in descending order, i.e., $q_1 \ge q_2 \ge \cdots \ge q_n$. This can be achieved in time $O(n \log n)$ by sorting $\boldsymbol{q}$. First, we show that this projection preserves the order in $\boldsymbol{q}$. The following lemma is a generalization proved in [74] for the Euclidean distance.

Lemma 7 (Order Preserving Lemma)**.** *Let $\boldsymbol{p}^*$ be the projection of $\boldsymbol{q}$ onto P_n s.t. $q_1 \ge q_2 \ge \cdots \ge q_n$. Then, the projection $\boldsymbol{p}^*$ w.r.t. any uniformly separable Bregman divergences also satisfies $p_1^* \ge p_2^* \ge ... \ge p_n^*$. In particular, $p_i = p_j$ if $q_i = q_j$.*

Proof. Assume that the lemma is false. Then, there exists a pair i and j such that $q_i \ge q_j$ and $p_i^* < p_j^*$. Let $\boldsymbol{p}'$ be the vector obtained by letting $p'_i = p_j^*$, $p'_j = p_i^*$, and $p'_k = p_k^*$ for $k \ne i, j$. It

can be easily verified that $\boldsymbol{p} \in P_n$. Now, observe that

$$\begin{aligned}
&\Delta_\Phi(\boldsymbol{p}^*, \boldsymbol{q}) - \Delta_\Phi(\boldsymbol{p}', \boldsymbol{q}) \\
&= \Phi(\boldsymbol{p}^*) - \Phi(\boldsymbol{p}') - \nabla\Phi(\boldsymbol{q}) \cdot (\boldsymbol{p}' - \boldsymbol{p}^*) \\
&= -(\nabla\phi(q_i)(p_i^* - p_j^*) - \nabla\phi(q_j)(p_j^* - p_i^*)) \\
&= (p_j^* - p_i^*)(\nabla\phi(q_i) + \nabla\phi(q_j)) > 0,
\end{aligned}$$

where the strict inequality holds since the assumption that $p_j^* > p_i^*$ and $\nabla\phi$ is strictly increasing function since ϕ is strictly convex. Therefore, this observation contradicts that $\boldsymbol{p}^*$ is the projection. Thus, the order is preserved in the projection. The equality is also preserved since if $q_i = q_j$, the projection preserves $p_i \geq p_j$ and $p_j \geq p_i$. □

Now we are ready to show one of our main technical lemmas.

Lemma 8. *For any* $\boldsymbol{q} \in \mathrm{Precons}(\mathcal{A})$*,*

$$\arg\min_{\boldsymbol{p} \in P_n} \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) = \arg\min_{\boldsymbol{p} \in P_n \cap \mathrm{Precons}(\mathcal{A})} \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}).$$

Proof. Let $\boldsymbol{p}^* = \arg\min_{\boldsymbol{p} \in P_n} \Delta_\Phi(\boldsymbol{p}, \boldsymbol{q})$. By definition of the projection, for any $\boldsymbol{p} \in P_n \cap \mathrm{Precons}(\mathcal{A}) \subseteq P_n$, $\Delta_\Phi(\boldsymbol{p}, \boldsymbol{q}) \geq \Delta_\Phi(\boldsymbol{p}^*, \boldsymbol{q})$. Further, by Lemma 7, $\boldsymbol{p}^*$ preserves the order and equality in $\boldsymbol{q}$. That is, $\boldsymbol{p}^*$ also satisfies the constraints defined by $\mathrm{Precons}(\mathcal{A})$. Therefore, we have $\boldsymbol{p}^* \in \mathrm{Precons}(\mathcal{A})$. These facts imply that $\boldsymbol{p}^*$ is indeed the projection of $\boldsymbol{q}$ onto $P_n \cap \mathrm{Precons}(\mathcal{A})$. □

So, by Lemma 6, when a vector $\boldsymbol{q} \in \mathrm{Precons}(\mathcal{A})$ is given, we can compute the projection of $\boldsymbol{q}$ onto $P_n \cap \mathrm{Precons}(\mathcal{A})$ by computing the projection of $\boldsymbol{q}$ onto P_n only. By applying the method of Lim et al. [46], the projection problem can be reduced to the following Isotonic Optimization problem.

$$\begin{aligned}
\arg\min_{\boldsymbol{y} \in \mathbb{R}^n} &\sum_{i=1}^{n} (d_i^*(y_i) - y_i c_i) \\
\text{sub. to:} &\; y_1 \leq y_2 \cdots \leq y_n
\end{aligned}$$

where $d_i(x_i) = \Delta_\phi(x_i, z_i)$ and d_i^* is the Legendre-Fenchel dual of d_i.

Lemma 9 (Cf. [46])**.** *The projection problem onto* P_n *is reducible to the following Isotonic Optimization problem.*

Proof. Let F denote the feasible set in the Isotonic Optimization problem. We have $d_i(x_i) = \max_{y_i}(y_i x_i - d_i^*(y_i))$. We can rewrite the problem as

$$\begin{aligned}
\min_{\boldsymbol{x}\in F} \sum_{i=1}^{n} &= \min_{\boldsymbol{x}\in F}(\sum_{i=1}^{n} \max_{y_i}(y_i x_i - d_i^*(y_i))) \\
&= \min_{\boldsymbol{x}\in F} \max_{y}(\sum_{i=1}^{n}(y_i x_i - d_i^*(x_i^*))) \\
&= \max_{y} \min_{\boldsymbol{x}\in F}(\sum_{i=1}^{n}(y_i x_i - d_i^*(y_i))) \\
&= \max_{y}(\sum_{i=1}^{n}(-d_i^*(y_i)) + \min_{\boldsymbol{x}\in F} \boldsymbol{y}\cdot\boldsymbol{x}).
\end{aligned}$$

Let us focus on the $\min_{\boldsymbol{x}\in F} \boldsymbol{y}\cdot\boldsymbol{x}$ term. If we let $Y_i = y_i - y_{i+1}$ for $i \in [n-1]$ and $Y_n = y_n$, we have $y_i = \sum_{l=i}^{n} Y_k$. This gives us

$$\begin{aligned}
\boldsymbol{y}\cdot\boldsymbol{x} &= \boldsymbol{y}\cdot\boldsymbol{c} + \boldsymbol{y}\cdot(\boldsymbol{x}-\boldsymbol{c}) \\
&= \boldsymbol{y}\cdot\boldsymbol{c} + \sum_{i=1}^{n} y_i(x_i - c_i) \\
&= \boldsymbol{y}\cdot\boldsymbol{c} + \sum_{i=1}^{n}(\sum_{k=i}^{n} Y_k)(x_i - c_i) \\
&= \boldsymbol{y}\cdot\boldsymbol{c} + \sum_{i=1}^{n}(\sum_{i=1}^{k}(x_i - c_i))Y_k
\end{aligned}$$

If any Y_k is larger than 0 for any $k \in [n-1]$, then $\inf_{\boldsymbol{x}} \boldsymbol{y}\cdot\boldsymbol{x} = -\infty$; we can set $x_i = c_i$ for $i \notin \{k, k+1\}$, $x_k \to -\infty$ and $x_{k+1} = c_k + c_{k+1} - x_k$. This means that we require $Y_k \leq 0$ for all k (i.e. $y_{i+1} \geq y_i$). So $\min_{\boldsymbol{x}} \boldsymbol{y}\cdot\boldsymbol{x} = \boldsymbol{y}\cdot\boldsymbol{c}$, obtained by setting $x_i = c_i$ for all i.

From a dual solution $\boldsymbol{y}$, we can recover $\boldsymbol{x}$ via the following relation: $\nabla\phi(x_i) = y_i + \nabla\phi(z_i)$

□

Isotonic Optimization problem, which has been studied in greater generality by Best et al. and Ahuja et al. The problem can be solved in $O(n \log n)$.

Theorem 16. *There exists an algorithm with input $\boldsymbol{q} \in \mathrm{Precons}(\mathcal{A})$ that outputs the projection of $\boldsymbol{q}$ onto $P_n \cap \mathrm{Precons}(\mathcal{A})$ in time $O(n \log n)$.*

Algorithm 9 Metarounding

Input: $\boldsymbol{p} \in P_n \cap \mathrm{Precons}(\mathcal{A})$ satisfying $p_1 \geq p_2 \geq \cdots \geq p_n$ and the transitive closure $\mathcal{A}^*$ of $\mathcal{A}$

Output: Permutation $\boldsymbol{\sigma} \in S_n \cap \mathrm{Precons}(\mathcal{A})$

1. Sort elements of $\boldsymbol{p}$ in the descending order, where for elements i, j such that $p_i = p_j$, i is larger than j if $(i, j) \in \mathcal{A}^*$, otherwise break the tie arbitrarily.
2. Output the permutation $\boldsymbol{\sigma}$ s.t. $\sigma_i = (n+1) - r_i$, where r_i is the ordinal of i in the above order.

4.3.3 Metarounding

Algorithm 9 is an implementation of our metarounding. The algorithm is quite simple. Roughly speaking, if the input $\boldsymbol{p} \in P_n \cap \mathrm{Precons}(\mathcal{A})$ is sorted as $p_1 \geq \cdots \geq p_n$, the algorithm outputs $\boldsymbol{\sigma} \in S_n$ such that $\sigma_1 \geq \cdots \geq \sigma_n$, i.e., $\boldsymbol{\sigma} = (n, n-1, \ldots, 1)$. Note that we need to break ties in $\boldsymbol{p}$ to construct $\boldsymbol{\sigma}$. Let $\mathcal{A}^*$ be the transitive closure of $\mathcal{A}$. Then, given an equivalence set $\{j \mid p_i = p_j\}$, we break ties so that if $(i, j) \in \mathcal{A}^*$, $\sigma_i \geq \sigma_j$. This can be done by, e.g., quicksort. First, we will show that the procedure guarantees that for each $i \in [n]$, $\sigma_i \leq (2 - 2/(n+1))p_i$, and then discuss its time complexity.

We prove the following lemma for Metarounding.

Lemma 10. *For any $\boldsymbol{p} \in P_n \cap \mathrm{Precons}(\mathcal{A})$ s.t. $p_1 \geq \cdots \geq p_n$, given $\boldsymbol{p}$, the output $\boldsymbol{\sigma}$ of Metarounding satisfies that for each $i \in [n]$, $\sigma_i \leq (2 - 2/(n+1))p_i$.*

Proof. For each $i \in [n]$, by definition of the permutahedron, we have

$$\sum_{j=1}^{i} p_j \leq \sum_{j=1}^{i-1} j = \frac{i(i-1)}{2}. \tag{4.2}$$

By the assumption that $p_1 \geq \cdots \geq p_n$, the average of $p_i + p_{i+1} + \cdots + p_n$ is not larger than p_i. Thus, we have

$$\begin{aligned} p_i &\geq \frac{\sum_{j=i}^{n} p_j}{n+1-i} = \frac{\sum_{j=1}^{n} p_j - \sum_{j=1}^{i-1} p_j}{n+1-i} \\ &\geq \frac{(n+i)(n+1-i)}{2(n+1-i)} = \frac{n+i}{2}, \end{aligned}$$

where the second inequality follows from (4.2). Thus, for each $i \in [n]$,

$$\frac{\sigma_i}{p_i} \leq \frac{n-i+1}{\frac{1}{2}(n+i)} = \frac{2(n+i-1)}{n+i} = 2 - \frac{4i-2}{n+i}.$$

Here, the second term $\frac{4i-2}{n+i}$ is minimized when $i = 1$. Therefore, $\sigma_i/p_i \leq 2 - 2/(n+1)$, as claimed. □

For computing Metarounding, we need to construct the transitive closure $\mathcal{A}^*$ of $\mathcal{A}$ before the protocol begins. It is well known that a transitive closure can be computed by using algorithms for all-pairs shortest paths. For this problem, the Floyd-Warshall algorithm can be used; it runs in time $O(n^3)$ and space $O(n^2)$ (see, e.g., [16]). When $\mathcal{A}$ is small, for example, $m << n^2$, we can use Johnson's algorithm, which runs in time $O(n^2 \log n + nm)$ and space $O(m^2)$.

The time complexity of Metarounding is $O(n^2)$, which is due to the sorting. The space complexity is $O(n^2)$, if we use the Floyd-Warshall algorithm with an adjacency matrix. The space complexity can be reduced to $O(m^2)$ if we employ Johnson's algorithm, which uses an adjacency list. On the other hand, we need an extra $O(\log m)$ factor in the time complexity since we need $O(\log m)$ time to check if $(i, j) \in \mathcal{A}^*$ when $\mathcal{A}^*$ is given as an adjacency list.

4.3.4 Main Result

Finally, by combining the discussions in previous subsections, we obtains the following result.

Theorem 17. *There exists an online linear optimization algorithm over $P_n \cap \mathrm{Precons}(\mathcal{A})$ such that*

1. *its $(2 - 2/(n+1))$-regret is $O(n^2\sqrt{T})$, and*
2. *its per-trial running time is $O(n^4)$.*

Note that the running time at each trial is dominated by the projection onto $\mathrm{Precons}(\mathcal{A})$, which takes time $O(n^4)$.

4.4 Lower Bound

In this section, we derive a lower bound for the regret for our online prediction problem over the permutahedron P_n. Here, we consider the special case of no precedence constraint being given.

Theorem 18. *For our prediction problem over the permutahedron P_n, the 1-regret is $\Omega(n^2\sqrt{T})$.*

Proof. We consider an adversary who makes random choices. More precisely, at each trial t, the adversary randomly chooses a loss vector $\boldsymbol{\ell}_t$ from $\boldsymbol{\ell}^0, \boldsymbol{\ell}^1$, where $\boldsymbol{\ell}^0$ ($\boldsymbol{\ell}^1$) is the loss vector in which the first $\frac{n}{2}$ elements are 0s (1s) and the remaining elements are 1s (0s). Then, for any online optimization algorithm that outputs $\boldsymbol{\sigma}_t \in S_t$ at trial t,

$$E[\sum_{t=1}^{T} \boldsymbol{\sigma}_t \cdot \boldsymbol{\ell}_t] = \frac{n(n+1)T}{4}.$$

Now, let us consider the best fixed permutation. Let $\boldsymbol{\sigma}^0 = (n, n-1, n-2, \ldots, 1)$ and $\boldsymbol{\sigma}^1 = (1, 2, 3, 4, \ldots, n)$. Suppose that $\boldsymbol{\ell}^0$ appears more frequently than $\boldsymbol{\ell}^1$ by k. The best permutation is $\boldsymbol{\sigma}^0$, and its cumulative loss is

$$\begin{aligned}
&\sum_{i=1}^{\frac{n}{2}} i\left(\frac{T}{2}+\frac{k}{2}\right) + \sum_{i=\frac{n}{2}+1}^{n} i\left(\frac{T}{2}-\frac{k}{2}\right) \\
=&\frac{n(n+1)T}{4} + \frac{k}{2}\left(2\frac{\frac{n}{2}(\frac{n}{2}+1)}{2} - \frac{n(n+1)}{2}\right) \\
=&\frac{n(n+1)T}{4} - \frac{k}{2}n\left(\frac{n+1}{2} - \frac{\frac{n}{2}+1}{2}\right) \\
=&\frac{n(n+1)T}{4} - \frac{k}{2}\frac{n^2}{4}.
\end{aligned}$$

The same argument follows for the opposite case, where $\boldsymbol{\ell}^1$ is more frequent by k. In fact, k can be expressed as $k = \sum_{t=1}^{T} \delta_t$, where each δ_t is a discrete uniform random variable that takes values of ± 1. Then, the expected regret of any online optimization algorithm is at least $\frac{n^2}{8} E\left[\left|\sum_{t=1}^{T} \delta_t\right|\right]$. By the central limit theorem, the distribution of $\sum_{t=1}^{T} \delta_t$ converges to a Gaussian distribution with mean 0 and variance $\sqrt{T}$. Thus, for sufficiently large T, $\Pr[|\sum_{t=1}^{T} \delta_t| \geq \sqrt{T}]$ is a constant: c $(0 < c < 1)$. Therefore, the expected regret bound has a lower bound of $\frac{n^2}{8} c\sqrt{T}$. This implies that there exists a sequence of loss vectors that enforces any online optimization algorithm to incur regret that is at least $\Omega(n^2\sqrt{T})$. □

In general, this lower bound on 1-regret is tight, since there are online algorithms that achieve a 1-regret with $O(n^2\sqrt{T})$ ([74, 2]).

It is natural to ask if the $(2 - 2/(n+1))$-regret $O(n^2\sqrt{T})$ is tight under precedence constraints. We do not yet have a lower bound for this case, but we will show that our algorithm is optimal unless there is an offline algorithm with an approximation ratio $\alpha < 2$.

Theorem 19. *If there exists a polynomial-time online linear optimization algorithm with an α-regret of $poly(n, m)\sqrt{T}$, then there also exists a randomized polynomial-time algorithm for the offline problem with an approximation ratio α.*

Proof. The proof is based on standard online-to-offline conversion methods that can be found in the online learning literature (see, e.g., [20]). Let A be such an online linear optimization algorithm, and let its output at each trial t be denoted as $\boldsymbol{\sigma}_t$. Let $\boldsymbol{\ell} \in [0, 1]^n$ be the loss vector in the offline problem. We consider an adversary who returns $\boldsymbol{\ell}_t = \boldsymbol{\ell}$ at each trial t. Then, the cumulative loss of A divided by T is bounded as follows:

$$\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\sigma}_t \cdot \boldsymbol{\ell} \leq \alpha \min_{\boldsymbol{\sigma} \in S_n \cap \mathrm{Precons}(\mathcal{A})} \boldsymbol{\sigma} \cdot \boldsymbol{\ell} + \frac{poly(n, m)}{T}.$$

Now, let $\widehat{\boldsymbol{\sigma}}$ be a uniformly and randomly chosen permutation from $\{\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_T\}$. Then,

$$E[\widehat{\boldsymbol{\sigma}} \cdot \boldsymbol{\ell}] \leq \alpha \min_{\boldsymbol{\sigma} \in S_n \cap \mathrm{Precons}(\mathcal{A})} \boldsymbol{\sigma} \cdot \boldsymbol{\ell} + \frac{poly(n, m)}{T}.$$

By setting $T = poly(n, m)$, the expected cumulative loss of $\widehat{\boldsymbol{\sigma}}$ is at most α times the cumulative loss of the best permutation (with a constant additive term), which completes the proof. □

4.5 Experiments

In this section, we show preliminary experiments with artificial data sets in order to compare the performance of our algorithm with other methods. The experiments were performed on a server with four cores of Intel Xeon CPU X5560 2.80 GHz and a memory of 198 GB. We implemented the programs using Matlab with its Optimization Toolbox. To generate the loss vector at each trial t, we independently and randomly specified each element $\ell_{t,i}$ of the loss vector $\boldsymbol{\ell}_t$ as follows: Let $\ell_{t,i} = 1$ with probability r_i and $\ell_{t,i} = 0$, otherwise. We set $r_i = i/n$ so that $E[\boldsymbol{\ell}_t] = (1/n, 2/n, ..., 1)$. We constructed random acyclic precedence constraints on n jobs in the following way. First, we constructed a random total order over n jobs (vertices). Then, we constructed an acyclic directed graph over n vertices by adding $\binom{n}{2}$ directed edges according to the total order. Finally, we kept each edge (i, j) alive with probability $\pi = 0.2$, and otherwise, we removed the edge. The resulting directed graph represented the set of precedence constraints.

Using the above method, for each fixed n and T, we constructed three random sequences

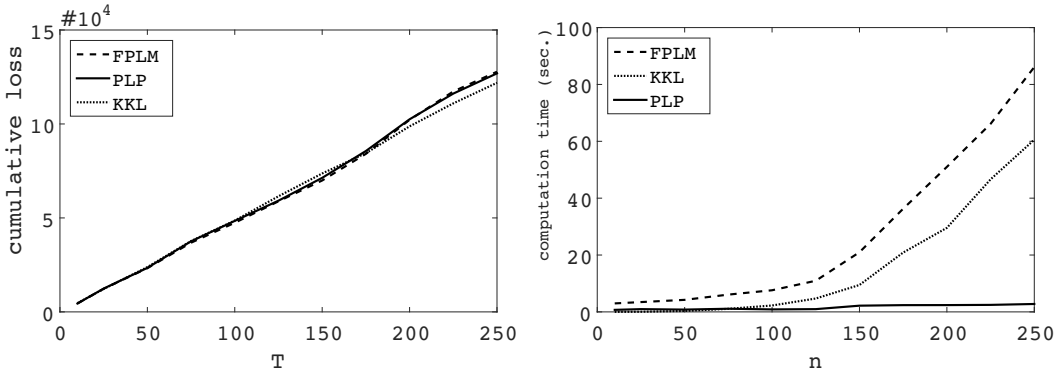

Figure 4.3: Upper panel: cumulative losses of the three algorithms with the artificial data set for $n = 50$ and $T = 10, 20, 50, 100, 200, 250$; lower panel: total computation times of the three algorithms with $T = 50$ with $n = 10, 20, 50, 100, 200, 250$.

of loss vectors and three random sets of precedence constraints. The results (cumulative loss or computation time) were then averaged.

We compared our algorithm PermLearnPrec (PLP) to the following algorithms. We used the offline-to-online conversion techniques of Kakade et al. [37] (KKL) and the metarounding technique of Fujita et al. [24] combined with (FPL) ([38]; FPLM). For the metarounding of Fujita et al., we set $\epsilon = 0.01$ to guarantee $(\alpha + \varepsilon)$-regret when using an α-approximation offline algorithm. We used the linear programming (LP) relaxation-based scheduling algorithm of Chudak and Hochbaum [14] as the offline algorithm. This algorithm solves a minimum-cut problem on a network with $O(n^2)$ nodes and $O(n^3)$ arcs. We used the maxflow algorithm of Boykov and Kolmogorov [7] to solve the minimum-cut problem. In our PLP algorithm, we solved the isotonic regression by using the standard quadratic programming (QP) solver in Matlab.

In Figure 4.3, we summarize the cumulative losses and total computation times, both averaged over three random data sets. As can be seen, the cumulative losses of all of the algorithms are quite similar. KKL performed slightly better than the other two algorithms, which is not surprising since they have almost identical α-regret bounds. On the other hand, if we consider the computation times of the algorithms, there is a very large difference. Our algorithm runs roughly 20 to 30 times faster than the other methods. The reason for this is that since the data are relatively "easy," the best permutation might not change frequently over time. Thus, in many trials, the projections onto the set of precedence constraints are already satisfied, and if this is the case, our algorithm can skip this step, whereas the other methods must compute the

precedence constraints for every trial.

4.6 Conclusion

In this thesis, we proposed a framework of the online combinatorial prediction with multiple projections. As an application of the framework, we showed a polynomial-time online linear optimization algorithm over the permutahedron under precedence constraints. Our algorithm achieves a $(2 - 2/(n + 1))$-regret bound $O(n^2\sqrt{T})$, which means that it can predict as well as the state-of-the art offline approximation algorithms in hindsight. The approximation algorithm for which the approximation ratio is strictly less than $2 - 2/(n + 1)$.

An interesting open question is how our online framework can be extended to minimize the sum of weighted completion times. We note that Woeginger [81] showed that the offline problem of minimizing the sum of weighted completion time can be reduced to that of minimizing the unweighted sum. This reduction might be useful for designing an online version.

Chapter 5

Boosting over non-deterministic ZDDs

5.1 Introduction

Most tasks in machine learning are formulated as optimization problems of various types. Recently, the amount of data to be treated is growing enormously large, and so the demands on scalable optimization methods are increasing. Probabilistic approach such as stochastic gradient descent methods [31] is now widely employed as standard techniques for large scale machine learning. Obviously, these methods require the time and/or the space complexity to be proportional to the size of given data.

In this thesis, we propose a new approach: *learning over compressed data*. That is, we first compress the given data somehow, and then employ various machine learning algorithms on the compressed data without explicitly re-constructing the original data. To be more precise, for any target machine learning algorithm to be employed, we apply an efficient algorithm running over the compressed data, which simulates the behavior of the target algorithm running over the original data, with the hope that the time and space complexity are significantly reduced when the data is well compressed. Although the complexity for compressing data of the first phase needs to be sufficiently small, we can expect great improvement of time and space complexity, especially when high compression ratio is achieved.

The methodology of working over compressed data has gained much attention in the areas of database and data mining, where various methods have been developed, say, for the string search from a compressed string and the frequent word extraction from compressed texts [45, 33, 28]. But, as far as the authors are aware, most of all the methods developed so far are limited to simple tasks such as search and counting, and few results are known for more complex tasks such as optimization in machine learning. Notable exceptions contain the results of Nishino et

al. [61] and Tabei et al. [75], respectively. Their methods use string compression techniques to perform matrix-based computations under small memory spaces. Our method, we will show later, is completely different from theirs.

As the first step toward establishing a general methodology of learning from compressed data, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data. The ZDD is a general data structure for representing a family of sets [54, 39], and is appropriate for our purpose. One of the reasons is that many results are reported in the literature that the ZDD indeed has ability of compactly representing various data in various domains [56, 55, 35].

In this thesis, we slightly generalize the ZDD by incorporating non-determinism and propose a new data structure called the non-deterministic ZDD (NZDD, for short). The NZDD has more flexibility for representing data because of the non-determinism. Also, our efficient simulation algorithms (showed later) fit naturally to the NZDDs. On the other hand, it seems to be NP-hard to construct an NZDD of minimal size from a given training data. An efficient construction method of succinct NZDDs is left as future work.

For the learning algorithm to be employed over the NZDD representation of the training data, we consider a boosting algorithm called the AdaBoost$*$ [66]. The AdaBoost$*$ is a refined version of the seminal boosting algorithm AdaBoost [19] and is guaranteed to find a hyperplane that maximizes the margin. In this thesis, we give an efficient implementation of the AdaBoost$*$. Its running time (per iteration) does not depend on the size of training data but is only linear in the size of the given NZDD. In addition, our proposed framework can be applicable to the AdaBoost as well and a similar guarantee also holds.

So, our method takes advantage when the size of NZDD is much smaller than the size of the training data, provided that the time complexity of constructing the NZDD is moderately small.

5.2 Problem statement and AdaBoost$*$

First we describe the problem of 1-norm hard margin maximization and then briefly review the AdaBoost$*$ which is one of the boosting algorithms that solve the problem.

5.2.1 1-norm hard margin maximization

Let X be a set called the instance space, and assume that we are given a finite set of *base hypotheses* $H = \{h_1, h_2, \ldots, h_n\} \subseteq \{h : X \to \{0, 1\}\}$. Note that the base hypotheses are usually assumed to take values in $\{-1, 1\}$, but since any function $g : X \to \{-1, 1\}$ can be represented as the difference of 0-1 valued functions (e.g., $g(x) = \mathbf{1}[g(x) = 1] - \mathbf{1}[g(x) = -1]$), we can assume 0-1 valued hypotheses without loss of generality. The base hypothesis class H defines a feature map, which maps any instance $x \in X$ to the feature vector $(h_1(x), h_2(x), \ldots, h_n(x))$ in the feature space $\{0, 1\}^n$. Later we will regard the feature vector for x as the set $H(x) = \{h_j \mid h_j(x) = 1\}$ and thanks to the assumption above, any base hypothesis $h_j \notin H(x)$ takes value 0 for x, which is a crucial property that makes our algorithm work.

Now we give the problem statement of 1-norm hard margin maximization. The input is a *sample* $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$, where x_i for $y_i = 1$ is called a positive instance and x_i for $y_i = -1$ a negative instance, and the output is a hyperplane in the feature space that separates the positive instances from the negative instances as much as possible. More precisely, the goal is to find

$$\boldsymbol{\alpha}^* = \arg \max_{\boldsymbol{\alpha} \in \{\mathbb{R}^n \mid \|\boldsymbol{\alpha}\|_1 = 1\}} \min_{1 \le i \le m} y_i \sum_{j=1}^{n} \alpha_j h_j(x_i). \tag{5.1}$$

We denote by $\boldsymbol{\alpha} \in \{\mathbb{R}^n \mid \|\boldsymbol{\alpha}\|_1 = 1\}$ the hyperplane whose normal vector is $\boldsymbol{\alpha}$, which also represents the convex combination of base hypotheses $f(x) = \sum_{j=1}^{n} \alpha_j h_j(x)$. Note that since the 1-norm of $\boldsymbol{\alpha}$ is normalized, $|f(x)|$ denotes the distance of the feature vector $(h_1(x), \ldots, h_n(x))$ to the hyperplane $\boldsymbol{\alpha}$ measured by ∞-norm. Thus, the signed distance $y_i f(x_i)$ (which is positive if and only if f correctly classifies x_i) is called the margin of the hyperplane $\boldsymbol{\alpha}$ with respect to the labeled instance (x_i, y_i). Let $\rho = \min_i y_i f(x_i)$ be the minimum margin of $\boldsymbol{\alpha}$ over all labeled instances in the sample. Note that $\boldsymbol{\alpha}^*$ is the hyperplane that maximizes ρ. It is well known that if $\rho > 0$, which means that the sample S is linearly separable, then the combined hypothesis f has a generalization error bound that is proportional to $1/\rho$ [51]. So, the goal of maximizing ρ is natural. Let $\rho^* = \min_i y_i \sum_j \alpha_j^* h_j(x_i)$ be the optimal margin. In what follows, we assume without loss of generality that all labeled feature vectors $(h_1(x_i), \ldots, h_n(x_i), y_i)$ are distinct.

5.2.2 AdaBoost$*$

The optimization problem (5.1) can be formulated as a linear programming problem of size $O(nm)$ and hence efficiently solved by an LP solver. However, in many cases, the number n of base hypotheses is very large (sometimes infinite), and thus the problem is infeasible for LP solvers. In such cases, boosting may provide an alternative way. In particular, the AdaBoost$*$ of Rätsch and Warmuth [66] provably converges to the maximum margin ρ^* within precision ν in $2\log(\frac{m}{\nu^2})$ iterations. Below we describe how the AdaBoost$*$ behaves when applied to the base hypothesis class H. On each round $t = 1, 2, \ldots, T$, it (i) computes a distribution d_t over the sample S, (ii) finds a base hypothesis $h_{j_t} \in H$ with the maximum *edge* (average margin) with respect to d_t, and (iii) updates the coefficient α_{j_t}. Finally, normalizing the coefficient $\boldsymbol{\alpha}$, it obtains a final hypothesis f. A pseudocode is given in Algorithm 10, where part (ii) above is implemented in a very naive manner: compute the edges of all base hypotheses (line 3-(a)) and then choose the maximum among them (line 3-(b)). So, this implementation is inefficient for a very large n. But, AdaBoost$*$ (and any other boosting algorithm) has a considerable advantage over LP solvers when the hypothesis class H has an efficient implementation, called the base learner, for this part: to find a base hypothesis with the maximum edge from a given distribution over the sample. In this case, the two lines (3-(a) and 3-(b)) are replaced by the base learner. The next theorem shows a performance guarantee of the AdaBoost$*$.

Theorem 20 (Rätsch and Warmuth [66])**.** *If* $T \geq \frac{2\log m}{\nu^2}$*, then AdaBoost*$*$ *(Algorithm 10) outputs a combined hypothesis* f *such that* $\min_{1\leq i\leq m} y_i f(x_i) \geq \rho^* - \nu$.

In this thesis, we consider the situation where the size n of H is small but the sample size m is very large, as is often the case, and both the direct applications of LP solvers and the AdaBoost$*$ may be useless.

5.2.3 AdaBoost

The AdaBoost, proposed by Freund and Schapire [19], is a precursor of the AdaBoost$*$. The algorithm, unlike the AdaBoost$*$, is not shown to provably maximize the hard margin. However, it is shown that it achieves at least half of the maximum hard margin asymptotically under weak technical conditions [65, 66]. Besides, the AdaBoost is much more popular because of its simplicity and the empirical performances. The behavior of the AdaBoost is almost the same as the AdaBoost$*$. More precisely, instead of 3. (c) and (d) in Algorithm 10, the AdaBoost updates

Algorithm 10 AdaBoost$*$

Input $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$
Output f

1. Let $\alpha_j = 0$ for $j = 1, \ldots, n$
2. Let $d_1(i) = 1/m$ for $i = 1, \ldots, m$
3. For $t = 1, \ldots, T$
   (a) Compute the edges
   $\gamma_{t,j} = \sum_{i=1}^{m} d_t(i) y_i h_j(x_i)$ for $j = 1, \ldots, n$.
   (b) Let $j_t = \arg\max_{1 \leq j \leq n} |\gamma_{t,j}|$; $\gamma_t = \gamma_{t,j_t}$.
   (c) Set $\rho_t = \min_{r=1,\ldots,t} |\gamma_r| - \nu$;
   (d) Update cofficients $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t} - \frac{1}{2} \log \frac{1+\rho_t}{1-\rho_t}$
   (e) Update weights
   $d_{t+1}(i) = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i))/Z_t$
   for $i = 1, \ldots, m$, where
   $Z_t = \sum_{i=1}^{m} d_t(i) \exp\left(-\alpha_{j_t} y_i h_{j_t}(x_i)\right)$
4. Let $f(x) = \sum_{j=1}^{n} \frac{\alpha_j}{\|\alpha\|_1} h_j(x)$

the coefficient as $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t}$. Therefore, the theoretical results we will show also are applicable to the AdaBoost.

5.3 A dag representation for samples

As a data structure for storing the sample, we propose a dag representation for a family of sets called the non-deterministic ZDD (NZDD, for short). It can be seen as a generalization of the ZDD by incorporating non-determinism.

5.3.1 Zero-suppressed decision diagram(ZDD)

A ZDD is defined by a 4-tuple $G = (V, E, \mathbb{N}, \Phi)$, where (V, E) is a directed acyclic graph and $\Phi : V \to \mathbb{N}$ is a function that assigns to each node v a natural number. Furthermore we require the additional definition as described below.

1. A leaf node is either the special node $\top$, or the special node $\bot$.

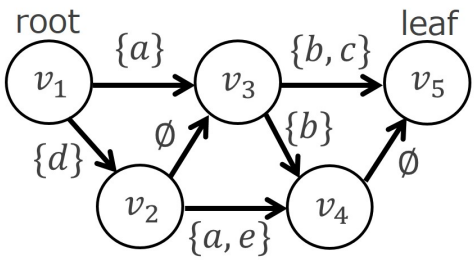

Figure 5.1: An NZDD representation for $\{\{a,b\},\{a,b,c\},\{a,d,e\},\{b,c,d\},\{b,d\}\}$

2. Each nornterminal node satisfies the following conditions: (i) The node has outdegree 2. One of the outgoing edges is named "LO" and the other is named "HI". In diagrams, we draw dotted lines for LO edges and solid lines for HI edges. (ii) A destination node is either terminal or labelled with an integer strictly larger than v. (iii) Thus we can omit arrowheads in diagrams because the edge directions can be inferred from the labels. (iv)The HI edge never points to the $\perp$ node.

3. There is exactly one node with zero indegree, which we call the root node. The above implies the root node is either terminal or labelled by the smallest integer in the dag.

4. If two nodes have the same label, then their LO or HI edges point to different nodes. In other words, there are no redundant nodes.

5.3.2 Non-deterministic ZDD (NZDD)

An NZDD is specified by a 4-tuple $G = (V, E, \Sigma, \Phi)$, where (V, E) is a directed acyclic graph with a single root and a single leaf , Σ is a ground set, and $\Phi : E \to 2^{\Sigma}$ is a function that assigns to each edge e a subset $\Phi(e)$ of Σ. Note that $\Phi(e)$ can be the empty set $\emptyset$. Furthermore we require the additional properies as described below. Let $\mathcal{P}_G$ be the set of all paths from the root to the leaf in G, where a path P in $\mathcal{P}_G$ is specified by the set of edges in P, i.e., $P \subseteq E$.

1. Every path $P \in \mathcal{P}_G$ represents a subset $S(P) \subseteq \Sigma$ defined as $S(P) = \bigcup_{e \in P} \Phi(e)$. Thus, the NZDD G defines a subset family as $L(G) = \{S(P) \mid P \in \mathcal{P}_G\} \subseteq 2^{\Sigma}$.

2. For every pair of paths $P, P' \in \mathcal{P}_G$, $S(P) \neq S(P')$ if $P \neq P'$.

3. For every path $P \in \mathcal{P}_G$, $\Phi(e) \cap \Phi(e') = \emptyset$ for any $e, e' \in P$ with $e \neq e'$.

Note that by the second property, there exists a one-to-one correspondence between the set of paths $\mathcal{P}_G$ and the subset family $L(G)$. In particular, we have $|\mathcal{P}_G| = |L(G)|$. The third property says that every element $a \in \Sigma$ appears at most once in every path $P \in \mathcal{P}_G$. That is, letting $E(a) = \{e \in E \mid a \in \Phi(e)\}$, we have $|E(a) \cap P| \leq 1$ for every $P \in \mathcal{P}_G$. Finally, we define the size of G as $|G| = \sum_{e \in E} |\Phi(e)|$. Note that the size $|G|$ can be significantly small as compared with the number of paths $|\mathcal{P}_G|$. In other words, the NZDD G is a compact representation for the subset family $L(G)$. As an example, we give in Fig. 5.2 an NZDD that represents a subset family.

5.3.3 NZDD representation for the sample

Now we describe how we represent the sample S as an NZDD.

Recall that $H(x) = \{h_j \in H \mid h_j(x) = 1\}$ for each instance $x \in X$. Let $Z^+ = \{H(x_i) \mid (x_i, 1) \in S\}$ and $Z^- = \{H(x_i) \mid (x_i, -1) \in S\}$ be the subset families with the ground set $\Sigma = H$, which correspond to the positive and the negative instances in the sample S, respectively. Let G^+ and G^- be NZDDs for the families Z^+ and Z^-, respectively. That is, $L(G^+) = Z^+$ and $L(G^-) = Z^-$. Finally, the NZDD G for the sample S is obtained by (i) putting an additional node as the global root with two outgoing edges labeled with $\emptyset$, where one edge is connected to the root of G^+ and the other is to the root of G^-, and (ii) merging the leaves of G^+ and G^- to a single leaf (See Fig. 5.3 for example). Note that G is not necessarily a minimal NZDD even if G^+ and G^- are minimal, because G may be further simplified by merging a node in G^+ and a node in G^-. But, we define G in this way, so that any path in G^+ and any path in G^- are disjoint.

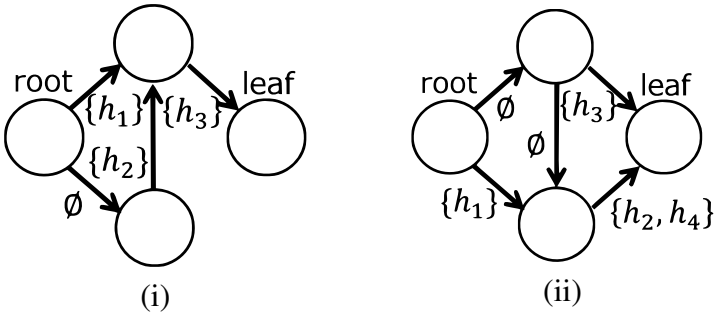

Figure 5.2: (i) An NZDD G^+ for $Z^+ = \{\{h_1, h_3\}, \{h_2, h_3\}\}$; (ii) An NZDD G^- for $Z^- = \{\{h_1, h_2, h_4\}, \{h_2, h_4\}, \{h_3\}\}$

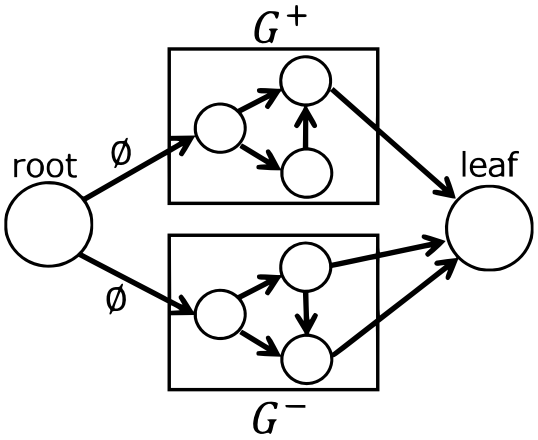

Figure 5.3: An NZDD for the sample consisting of positive instances Z^+ and negative instances Z^-

5.3.4 Relations to ZDDs and NFAs

We show that the ZDD representation is a special case of the NZDD representation. To see this, we consider the class of NZDDs of the following form:

1. Each edge e is labeled with either a singleton or the empty set. That is, $|\Phi(e)| \leq 1$.
2. Each internal node has one or two outgoing edges. If it has two outgoing edges, one of them is labeled with the empty set.
3. There exists a fixed ordering over Σ such that for any pair of edges e and e' labeled with singletons $\{a\}$ and $\{a'\}$, respectively, if e is an ancestor of e', then a precedes a' in this ordering.

It is easy to see that any ZDD can be seen as an NZDD in this form.

Conversely, consider the class of NZDDs of the following form:

1. It is ordered. That is, the third condition above is satisfied.
2. For each pair of edges e and e' outgoing from a common node, $\Phi(e) \cap \Phi(e') = \emptyset$.

Then, we can show that any NZDD of this class has an equivalent ZDD of the same size. So, only the difference of ordered NZDDs from ZDDs is that we allow non-determinism, i.e., $\Phi(e) \cap \Phi(e') \neq \emptyset$.

Next we consider the relation of ordered NZDDs to NFAs. Under the ordering over Σ, we can identify a subset $\{a_1, a_2, \ldots, a_k\} \subseteq \Sigma$ with a string $a_1 a_2 \cdots a_k \in \Sigma^*$ over the alphabet Σ,

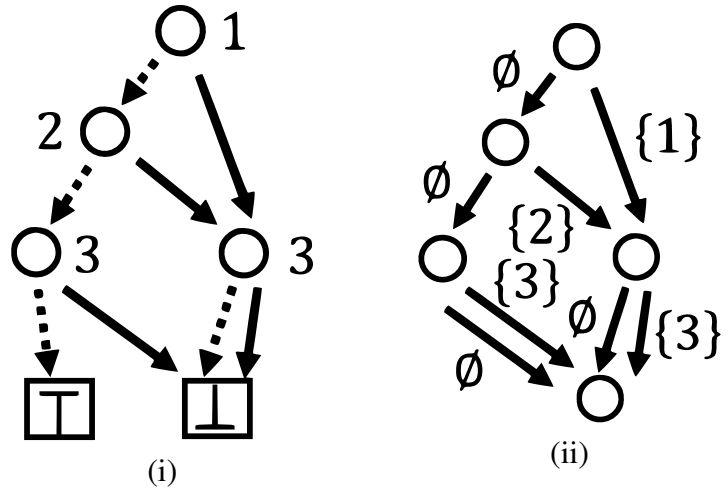

Figure 5.4: (i) An ZDD for $\{\{1\},\{2\},\{3\},\{1,3\},\{2,3\}\}$; (ii) A NZDD corresponding to the ZDD.

where $a_1 < a_2 < \cdots < a_k$ under the ordering $<$. Note that the empty set corresponds to the empty string ϵ. In this way, a subset family can be seen as a language. From this viewpoint, we can regard an NZDD G as an NFA that recognizes the language $L(G)$, with the root identified with the start state and the leaf with the unique accepting state. The difference is that, in the NZDD representation, we have only a single accepting path for each string in the language. This implies that any DFA for such a language can be converted to an NZDD in an obvious way. Note that in order to make the accepting state unique, we may need to put an additional leaf and connecting every accepting state to the leaf by an additional edge labeled with the empty set (ϵ-transition).

5.3.5 Complexity of constructing NZDDs

When given a subset family $L \subseteq 2^{\Sigma}$, we want to compute a minimal NZDD G with $L(G) = L$. So far, the time complexity of the problem is unknown, but it seems to be NP-hard because so are the closely related problems, namely, construction of a minimal ZDD (over all ordering) [39] and construction of a minimal NFA [36]. On the other hand, we have a polynomial time algorithm for constructing a minimal ZDD when given an ordering [67] and a linear time algorithm for constructing a minimal DFA for a finite lanugage [67]. So, practically, we can use these algorithms for constructing an ordered NZDD of small size.

5.4 Simulating AdaBoost$*$ over an NZDD representation for the sample

In this section, we give an algorithm that efficiently simulates the AdaBoost$*$ over an NZDD G that represents a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, without explicitly reconstructing the sample S from G. In particular, the running time (per iteration) of our algorithm does not depend on the sample size m but is linear in the size of G. First we state the main theorem.

Theorem 21. *There exists an algorithm that, when given an NZDD G that represents a sample S, exactly simulates AdaBoost$*$ whose running time is $O(|G|)$ per iteration.*

So, if the sample is significantly compressed in the NZDD representation, our algorithm runs much faster than the direct application of the AdaBoost$*$ when the computation time of constructing G from S is negligible. More specifically, if we use a linear time algorithm for constructing an NZDD from a minimal DFA as described in the previous section, then the total running time of our algorithm is $O(nm + T|G|)$, whereas the total running time of the direct application of the AdaBoost$*$ is $O(nmT)$. So, if $|G| \ll nm$, then our algorithm would be faster[1].

Further, since the AdaBoost is almost identical to the AdaBoost$*$ in an algorithmic sense, we have the following corollary as well.

Corollary 22. *There exists an algorithm that, when given an NZDD G representing S, simulates AdaBoost whose running time is $O(|G|)$ per iteration.*

Below we describe a basic idea of the algorithm. Obviously, we cannot explicitly maintain the distribution d_t over the sample S. Instead, we maintain one weight $w_{t,e}$ for each edge e of G, so that the edge weights w_t implicitly represents d_t. The same idea is used in [76] to efficiently simulate online prediction algorithms with multiplicative update rules, where the decision space is the set of paths of a given directed acyclic graph.

To describe the idea formally, we need some additional notations. Recall that there exists a one-to-one correspondence between the sample S and the set of all root-to-leaf paths $\mathcal{P}_G$ in G. So, we identify a labeled instance $(x_i, y_i) \in S$ with a path $P \in \mathcal{P}_G$, and we will denote the weight for the instance by $d_t(P)$ instead of $d_t(i)$. Furthermore, let $\mathcal{P}_G^+$ and $\mathcal{P}_G^-$ denote the set of paths that pass through G^+ and the set of paths that pass through G^-, respectively.

[1] Note that it always holds that $|G| \leq nm$.

Now we give the two conditions C1 and C2 that the edge weights w_t need to satisfy, so as to represent the path distribution d_t.

C1. The edge weights w_t need to satisfy

$$d_t(P) = \prod_{e \in P} w_{t,e}$$

for every path (labeled instance) $P \in \mathcal{P}_G$.

C2. The outflow from each internal node should be one. That is, w_t need to satisfy

$$\sum_{a:(u,a)\in E(G)} w_{t,(u,a)} = 1$$

for every internal node u, where $E(G)$ denotes the set of edges of G.

What we need to show is how to simulate AdaBoost$*$ efficiently by using the edge weights w_t. More precisely, we need to simulate the two parts of AdaBoost$*$:

(a) updating the path distributions d_t (corresponding to Line 2 and Line 3-(e) of Algorithm 1), and

(b) computing the edges $\gamma_{t,j}$ (corresponding to Line 3-(a)).

In the following subsections, we give algorithms that simulate the two parts.

5.4.1 Updating the path distributions d_t

To simulate this part, we use the Weight Pushing algorithm developed by [57], which rearranges the edge weights so that relative weights on the path remain unchanged but again satisfy the two conditions. More precisely, the Weight Pushing algorithm has the following property.

Proposition 23 (Mohri [57])**.** *When given arbitrary edge weights $w'_e \geq 0$, the Weight Pushing algorithm produces edge weights w_e in time $O(|E|)$ such that w_e satisfies condition C2 and*

$$\prod_{e \in P} w_e = \frac{\prod_{e \in P} w'_e}{\sum_{P \in \mathcal{P}_G} \prod_{e \in P} w'_e}$$

for every path $P \in \mathcal{P}_G$.

Algorithm 11 Initializing the path distribution

1. Let $w'_e = 1$ for all edges in G.
2. Apply the Weight Pushing algorithm to w' and get w_1.

Algorithm 12 Updating the path distribution

1. Forall $e \in E(G)$, let $w'_e = w_{t,e}$
2. Forall $e \in E(G^+)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(-\alpha_{j_t})$
3. Forall $e \in E(G^-)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(\alpha_{j_t})$
4. Apply the Weight Pushing algorithm to w' and get w_{t+1}.

The initialization of the path weights ($d_1(P) = 1/m$) of Line 2 of Algorithm 10 can be realized by the two steps as described in Algorithm 11. It is justified by Proposition 23 which implies

$$\prod_{e \in P} w_{1,e} = \frac{1}{|\mathcal{P}_G|} = 1/m = d_1(P).$$

Moreover, the running time of Algorithm 11 is $O(|E|)$.

The update of path distributions of Line 3-(e) of Algorithm 10 can be realized by multiplying the weights of the edges e such that $h_{j_t} \in \Phi(e)$, and applying the Weight Pushing algorithm. See Algorithm 12 for more details.

Below we give a justification of Algorithm 12.

Lemma 11. *Algorithm 12 exactly simulates Line 3-(e) of Algorithm 10 in time $O(|E|)$.*

Proof. Let P be a path in $\mathcal{P}_G$ that corresponds to a labeled instance (x_i, y_i) and examine the quantity $\prod_{e \in P} w'_e$. Recall that

$$\bigcup_{e \in P} \Phi(e) = \{h_j \in H \mid h_j(x_i) = 1\}$$

by the definition of the NZDD construction for S.

First consider the case where $h_{j_t}(x_i) = 0$. In this case, there is no edge $e \in P$ such that

$h_{j_t} \in \Phi(e)$. Therefore, $w'_e = w_{t,e}$ for all edges e in P. Thus,

$$\begin{aligned}\prod_{e \in P} w'_e = \prod_{e \in P} w_{t,e} &= d_t(P) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

Next consider the case where $y_i = 1$ (i.e., P passes through G^+) and $h_{j_t}(x_i) = 1$. In this case, there exists a unique edge $e \in P$ such that $h_{j_t} \in \Phi(e)$. The uniqueness comes from Property 3 of the NZDD. So, $w'_e = w_{t,e} \exp(-\alpha_{j_t})$ for the edge e. Since $h_{j_t} \notin \Phi(e')$ for any other edge $e' \in P$, we have

$$\begin{aligned}\prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e} \right) \exp(-\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

For the last case where $y_i = -1$ and $h_{j_t}(x_i) = 1$, a similar argument to the case above gives

$$\begin{aligned}\prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e} \right) \exp(\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).\end{aligned}$$

Hence for all paths P, we have

$$\prod_{e \in P} w'_e = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).$$

Therefore, Proposition 23 ensures that w_{t+1} represents the path distribution d_{t+1} as desired. □

5.4.2 Computing the edges $\gamma_{t,j}$

To compute $\gamma_{t,j}$, we first compute the following quantity

$$f_e = \sum_{P \in \mathcal{P}_G : e \in P} d_t(P)$$

for all edges e, which can be interpreted as the probability flow of edge e, i.e., the probability that the path P goes through edge e when P is chosen according to the distribution d_t. Since G is a directed acyclic graph, we can compute f_e for all edges e by dynamic programming (e.g., the forward-backward algorithm) in linear time. Then, it is not hard to see that $\gamma_{t,j}$ can be computed by

$$\gamma_{t,j} = \sum_{e \in E(G^+): h_j \in \Phi(e)} f_e - \sum_{e \in E(G^-): h_j \in \Phi(e)} f_e.$$

We summarize the result as in the following lemma.

Lemma 12. *There exists an algorithm that exactly simulates Line 3-(a) of Algorithm 10 in time* $O(|G|)$.

Theorem 21 follows from Lemma 11 and Lemma 12.

5.5 Conclusions

We have proposed the NZDD, a variant of ZDDs for representing the training data succinctly and algorithms, given a NZDD, simulate AdaBoost$*$ as well as AdaBoost on the training data efficiently. As future work, we will evaluate empirical performances of our method on real and synthetic data sets. Also, investigation of efficient construction methods of NZDDs is important. One of open problems is to extend our method to the 1-norm soft margin maximization, where additional constraints make the theoretical results of the direct application of our method worse. Also, the problem of obtaining similar results for the 2-norm support vector machines remains open.

Chapter 6

Conclusion

In this thesis, we considered online prediction for combinatorial concept classes and learning over compressed data.

In Chapter 3, we considered combinatorial online optimization problems and proposed a new method to construct a low-regret online algorithm by using an offline algorithm. The method was that if the corresponding offline algorithm is based on solving a relaxation problem of original and base on metarounding, an efficient online algorithm can be constructed. Furthermore, we showed that the problem of metarounding has similar structure to boosting and solved metarounding by using boosting-like algorithm.

In Chapter 4, we extended our construction method for combinatorial online optimization. In our previous method, we assumed existing efficient algorithms for the projection and the metarounding. For this assumption, we generalizes the projection to multiple projections. Further, we exhibited a case where the multiple projection are useful in a problem of online scheduling of jobs with a single machine under precedence constraints. We proposed a polynomial time online scheduling algorithm, which have a tight cumulative loss bound.

In Chapter 5, we considered efficient learning methods over compressed data sets. We employed non deterministic ZDDs as the data structure for representing the data sets. For the learning algorithm to be applied over the ZDDs, we consider a boosting algorithm called AdaBoost*. We gave an efficient implementation of AdaBoost* and its running time does not depend on the size of data sets but is linear in the size of the ZDD. Our method took advantage when the size of the ZDD is much smaller than the size of the training data and the time of constructing the ZDD is moderately small.

Bibliography

[1] N. Ailon. Aggregation of Partial Rankings, p-Ratings and Top-m Lists. *Algorithmica*, 57(2):284–300, 2008.

[2] N. Ailon. Improved bounds for online learning over the permutahedron and other ranking polytopes. In *Proceedings of 17th International Conference on Artificial Intelligence and Statistics (AISTAT2014)*, pages 29–37, 2014.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), 2008.

[4] C. Ambühl, M. Mastrolilli, N. Mutsanas, and O. Svensson. On the Approximability of Single-Machine Scheduling with Precedence Constraints Christoph Ambühl. *Mathematics of Operations Research*, 36(4):653–669, 2011.

[5] C. Ambühl, Monaldo Mastrolilli, and O. Swensson. Inapproximability Results for Sparsest Cut, Optimal Linear Arrangement, and Precedence Constrained Scheduling. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pages 329 – 337, 2007.

[6] J.-Y. Audibert, S. Bubeck, and G. Lugosi. Minimax Policies for Combinatorial Prediction Games. In *Proceedings of the 24th Annual Conference on Learning Theory (COLT 2011)*, pages 107–132, 2011.

[7] Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2004.

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

[9] D. R. Carr and S. Vempala. Randomized metarounding. *Random Structure and Algorithms*, 20(1):343–352, 2002.

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

[11] N. Cesa-Bianchi and G. Lugosi. Combinatorial Bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.

[12] C. Chekuri and R. Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[13] S. Chen, T. Lin, I. King, M. R. Lyu, and W. Chen. Combinatorial pure exploration of multi-armed bandits. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pages 379–387, 2014.

[14] F. A. Chudak and D. S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25:199–204, 1999.

[15] R. Combes, M. S. Talebi, A. Proutière, and M. Lelarge. Combinatorial bandits revisited. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pages 2116–2124, 2015.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and S. Clifford. *Introduction to Algorithms*. The MIT Press, third edit edition, 2009.

[17] J. R. Correa and A. S. Schulz. Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.

[18] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[19] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[20] Y. Freund and R. E. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–299, 1999.

[21] S. Fujishige. *Submodular functions and optimization*. Elsevier Science, 2nd edition, 2005.

[22] T. Fujita, K. Hatano, S. Kijima, and E. Takimoto. Online Linear Optimization for Job Scheduling under Precedence Concstraints. In *Proceedings of 26th International Conference on Algorithmic Learning Theory(ALT 2015)*, volume 6331 of *LNCS*, pages 345–359, 2015.

[23] T. Fujita, K. Hatano, S. Kijima, and E. Takimoto. Online Combinatorial Optimization with Multiple Projections and Its Application to Scheduling Problem. *IEICE Transaction*, 2018. to be published.

[24] T. Fujita, K. Hatano, and E. Takimoto. Combinatorial Online Prediction via Metarounding. In *Proceedings of 24th Annual Conference on Algorithmic Learning Theory (ALT 2013)*, pages 68–82, 2013.

[25] T. Fujita, K. Hatano, and E. Takimoto. Boosting over non-deterministic ZDDs. In *Proceedings of The 12th International Conference and Workshops on Algorithms and Computation (WALCOM 2018)*, 2018.

[26] D. Garber. Efficient online linear optimization with approximation algorithms. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 627–635, 2017.

[27] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[28] K. Goto, H. Bannai, S. Inenaga, and M. Takeda. Fast q-gram mining on SLP compressed strings. *Journal of Discrete Algorithms*, 18:89–99, 2013.

[29] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[30] E. Hazan. The convex optimization approach to regret minimization. Optimization for Machine Learning, 2011.

[31] E. Hazan. *Introduction to online convex optimization*. Now Publishers Inc., 2016.

[32] D. P. Helmbold and M. K. Warmuth. Learning Permutations with Exponential Weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009.

[33] D. Hermelin, G. M. Landau, S. Landau, and O. Weimann. A unified algorithm for accelerating edit-distance computation via text-compression. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, 2009.

[34] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[35] T. Inoue, K. Takano, T. Watanabe, J. Kawahara, R. Yoshinaka, A. Kishimoto, K. Tsuda, S.-i. Minato, and Y. Hayashi. Distribution Loss Minimization With Guaranteed Error Bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, 2014.

[36] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[37] S. Kakade, A. T. Kalai, and L. Ligett. Playing games with approximation algorithms. *SIAM Journal on Computing*, 39(3):1018–1106, 2009.

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

[39] D. E. Knuth. *Art of Computer Programming, Volume 4, Fascicle 1, The: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley, 2009.

[40] W. M. Koolen, M. K. Warmuth, and J. Kivinen. Hedging Structured Concepts. In *Proceedings of the 23rd Conference on Learning Theory*, pages 93–105, 2010.

[41] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvari. Combinatorial cascading bandits. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, 2015.

[42] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvári. Tight regret bounds for stochastic combinatorial semi-bandits. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, 2015.

[43] E. L. Lawler. On Sequencing jobs to minimize weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics 2*, 2:75–90, 1978.

[44] J. K. Lenstra and A. H. G. R. Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[45] Y. Lifshits. Processing compressed texts: a tractability border. In *CPM'07 Proceedings of the 18th annual conference on Combinatorial Pattern Matching*, pages 228–240, 2007.

[46] C. H. Lim and S. J. Wright. Efficient bregman projections onto the permutahedron and related polytopes. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1205–1213, 2016.

[47] T. Lin, B. D. Abrahao, R. D. Kleinberg, J. Lui, and W. Chen. Combinatorial partial monitoring game with linear feedback and its applications. In *Proceedings of the 31th International Conference on Machine Learning*, pages 901–909, 2014.

[48] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[49] R. Luss and S. Rosset. Generalized isotonic regression. *Journal of Computational and Graphical Statistics*, 23(1):192–210, 2014.

[50] R. Luss, S. Rosset, and M. Shahar. Efficient regularized isotonic regression with application to gene-gene interaction search. *Annals of Applied Statistics*, 6(1), 2012.

[51] O. L. Mangasarian. Arbitrary-norm separating plane. *Oper. Res. Lett.*, 24(1-2):15–23, 1999.

[52] F. Margot, M. Queyranne, and Y. Wang. Decompositions, Network Flows, and a Precedence Constrained Single-Machine Scheduling ProblemDecompositions, Network Flows, and a Precedence Constrained Single-Machine Scheduling Problem. *Operations Research*, 51(6):981–992, 2003.

[53] W. Maxwell and J. Muckstadt. Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research*, 33:1316–1341, 1985.

[54] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *DAC '93: Proceedings of the 30th international conference on Design automation*, 1993.

[55] S. Minato and T. Uno. Frequentness-transition queries for distinctive pattern mining from time-segmented databases. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM2010)*, pages 339–349, 2010.

[56] S. Minato, T. Uno, and H. Arimura. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 234–246, 2008.

[57] M. Mohri. General algebraic frameworks and algorithms for shortest-distance problems. Technical report, Technical Memorandum 981210-10TM, AT&T Labs- Research, 62 pages, 1998.

[58] H. R. Mohring, A. S. Schulz, and M. Uetz. Approximation in Stochastic Scheduling: The Power of LP-based Priority Policies. *Journal of the ACM*, 46(6):924–942, 1999.

[59] G. Neu. First-order regret bounds for combinatorial semi-bandits. In *Proceedings of The 28th Conference on Learning Theory, COLT*, pages 1360–1375, 2015.

[60] G. Neu and M. Valko. Online combinatorial optimization with stochastic decision sets and adversarial losses. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pages 2780–2788, 2014.

[61] M. Nishino, N. Yasuda, S.-i. Minato, and M. Nagata. Accelerating Graph Adjacency Matrix Multiplications with Adjacency Forest. In *Proceedings of the 2014 SIAM International Conference on Data Mining (SDM 14)*, pages 1073–1081, 2014.

[62] L. Qin, S. Chen, and X. Zhu. Contextual combinatorial bandit and its application on diversified online recommendation. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 461–469, 2014.

[63] H. Rahmanian and M. K. Warmuth. Online dynamic programming. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 2824–2834, 2017.

[64] A. Rajkumar and S. Agarwal. Online decision-making in general combinatorial spaces. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pages 3482–3490, 2014.

[65] G. Rätsch. *Robust Boosting via Convex Optimization: Theory and Applications*. PhD thesis, University of Potsdam, 2001.

[66] G. Rätsch and M. K. Warmuth. Efficient margin maximizing with boosting. *Journal of Machine Learning Research*, 6:2131–2152, 2005.

[67] D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189, 1992.

[68] R. E. Schapire and Y. Freund. *Boosting: Foundation and Algorithms*. MIT Press, 2012.

[69] A. Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

[70] A. S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds. In *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization (IPCO1996)*, pages 301–315, 1996.

[71] M. Skutella and M. Uetz. Stochastic Machine Scheduling with Precedence Constraints. *SIAM Journal on Computing*, 34(4):788–802, 2005.

[72] J. Spouge, H. Wan, and W. Wilbur. Least squares isotonic regression in two dimensions. *J. Optimization Theory and Apps.*, 117:585–605, 2003.

[73] A. Srinivasan. Improved approximations of packing and covering problems. In *27th ACM Symposium on the Theory of Computing*, pages 268–276, 1995.

[74] D. Suehiro, K. Hatano, S. Kijima, E. Takimoto, and N. Kiyohito. Online prediction under submodular constraints. In *In Proceedings of 23th Annual Conference on Learning Theory (ALT 2012)*, 2012.

[75] Y. Tabei, H. Saigo, Y. Yamanishi, and S. J. Puglisi. Scalable Partial Least Squares Regression on Grammar-Compressed Data Matrices. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 1875–1884, 2016.

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

[77] M. Warmuth, K. Glocer, and G. Rätsch. Boosting Algorithms for Maximizing the Soft Margin. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 1585–1592, 2007.

[78] M. K. Warmuth and D. Kuzmin. Randomized Online PCA Algorithms with Regret Bounds that are Logarithmic in the Dimension. *Journal of Machine Learning Research*, 9:2287–2320, 2008.

[79] Z. Wen, B. Kveton, and A. Ashkan. Efficient learning in large-scale combinatorial semi-bandits. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, pages 1113–1122, 2015.

[80] D. P. Williamson and D. B. Shmoys. *The Design of Approximation ALgorithms*. Cambridge University Press, 2011.

[81] G. J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*, pages 862–874, 2001.

[82] G. J. Woeginger and P. Schuurman. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 1999.

[83] S. Yasutake, K. Hatano, S. Kijima, E. Takimoto, and M. Takeda. Online Linear Optimization over Permutations. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC 2011)*, pages 534–543, 2011.

[84] S. Yasutake, K. Hatano, E. Takimoto, and M. Takeda. Online Rank Aggregation. In *Proceedings of 4th Asian Conference on Machine Learning (ACML2012)*, volume 25 of *JMLR W&CP*, pages 539–553, 2012.

[85] G. M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics 152. Springer-Verlag, 1995.

[86] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML '03)*, pages 928–936, 2003.