Combinatorial Online Prediction

Hatano, Kohei
Kyushu University

https://hdl.handle.net/2324/2320605
Combinatorial Online Prediction

Kohei Hatano
Kyushu University / RIKEN AIP
hatano@inf.kyushu-u.ac.jp

Abstract—We present a short survey on recent results on combinatorial online prediction in the adversarial setting.

Index Terms—online learning, combinatorial online prediction, online convex optimization, combinatorial optimization

I. INTRODUCTION

Combinatorial online prediction problems arise in many situations such as scheduling [13], ranking [20], [43], routing [5], [39], network optimization [26], and so on. Recent advances in online convex optimization, offline convex and discrete optimization, and discrete data structures enable us to investigate broader problems than ever. In this paper, we survey results in combinatorial online prediction and related areas, as well as their applications.

The combinatorial online decision-making problem is defined as the following protocol between the player and the adversary. Let $\mathcal{C} \subseteq \mathbb{R}_+^n$ be the set of decisions of interest. For each trial $t = 1, \ldots, T$,

1) the player chooses a decision $c_t \in \mathcal{C}$,
2) the adversary chooses a loss vector $\ell_t \in [0, 1]^n$, and
3) the adversary provides the player a feedback:
   - The feedback is the loss vector $\ell_t$ itself (full-information setting).
   - The feedback is the inner product $c_t \cdot \ell_t$ only (bandit setting).

There are options for feedbacks from the adversary. In the full-information setting, the loss vector $\ell_t$ is always revealed to the player. In the bandit setting, the loss, which is the inner product between the decision $c_t$ and the loss vector $\ell_t$, is provided to the learner, while the information of the loss vector is not given. There is an intermediate setting between these, called semi-bandit setting. In the semi-bandit setting, where typically $\mathcal{C} \subseteq \{0, 1\}^n$, not only the loss, but also components of the loss vector $\ell_{t,i}$ s.t. $c_{t,i} = 1$ are shown to the learner. This setting is natural in some applications, e.g., online routing where cost of each edge in the chosen route is available.

The goal of the player is to minimize the $\alpha$-regret for small $\alpha \geq 1$:

$$\alpha\text{-regret} = \mathbb{E} \left[ \sum_{t=1}^{T} \ell_t \right] - \alpha \min_{c \in \mathcal{C}} \sum_{t=1}^{T} c^* \cdot \ell_t,$$

where the expectation is taken w.r.t. the randomness of the player. In typical settings, we consider the cases with $\alpha = 1$ and 1-regret. For simplicity, we abbreviate 1-regret as just regret.

When the set $\mathcal{C}$ is finite and the size is small or $\mathcal{C}$ is a closed convex set, the problem is called the expert setting, or the online linear optimization, for which there are algorithms whose regret bounds are $O(\sqrt{T})$, which are optimal w.r.t. $T$ and running times at each trial are polynomial in $|\mathcal{C}|$ for the expert setting and polynomial in $n$ (other parameters are omitted).

However, when the size of $\mathcal{C}$ is huge, e.g., exponentially large w.r.t. $n$ or other relevant parameters, those optimal algorithms are inefficient. We will review three approaches for combinatorial online prediction problems.

Note that we will focus on the setting where the adversary is literally “adversarial”. More precisely, we assume that the adversary is oblivious, meaning that it determines loss vectors without seeing the player’s decisions before the game starts. This setting is not weak as it seems, since if the player is a deterministic algorithm, the adversary could know its behavior in advance. But, we assume that the player is a randomized algorithm. Further, the adversary is assumed to not see results of player’s random choices. There are more relaxed settings where the adversary is adaptive, in which it can see the random choices of the player. Some approaches work in the adaptive setting, but some do not.

Another relevant assumption would be that the adversary is random and the loss vectors follow some probability distribution. We omit this case and we recommend interested readers to see, e.g., the survey of Cesa-Bianchi and Lugosi [7] and Hazan [19], respectively.

II. PRELIMINARIES

We begin with some notations. For $n \geq 1$, let $[n] = \{1, \ldots, n\}$. For $n \geq 2$, let $\Delta_n$ be the probability simplex over $1, \ldots, n$, i.e., $\Delta_n = \{p \in [0, 1]^n \mid \sum_{i=1}^{n} p_i = 1\}$. For a finite set $\mathcal{C} \subseteq \mathbb{R}^n$, the convex hull $\text{conv}(\mathcal{C})$ is defined as $\text{conv}(\mathcal{C}) = \{\sum_{c \in \mathcal{C}} \alpha_c c \mid \sum_{c \in \mathcal{C}} \alpha_c = 1, \alpha \geq 0\}$.

A. Examples of combinatorial sets

We explain several combinatorial objects often treated in the literature.

a) $k$-sets: A $k$-set is a vector in $\{0, 1\}^n$ whose 1-components are exactly $k$ ($1 \leq k \leq n$). $k$-sets represent $k$ multiple choices from $n$ fixed alternatives. The set $\mathcal{C}$ of $k$-sets is represented as $\mathcal{C} = \{x \in \{0, 1\}^n \mid \sum_{i \in [n]} x_i = k\}$.

b) Paths in a Directed Acyclic Graph: Given a directed acyclic graph (DAG) with the source and sink nodes $G = (V, E)$ with $|V| = n$ and $|E| = m$, $\mathcal{C} = \{c \in \{0, 1\}^m \mid c$ represents a path from the source node to the sink node in $G\}$. 
c) Permutations: A permutation \( \pi \) over \([n]\) is a bijection from \([n]\) to \([n]\). Let \( S_n \) be the set of permutations over \([n]\).
Given a permutation \( \pi \in S_n \), the associated permutation matrix \( \Pi \in [0,1]^{n \times n} \) is defined as \( \Pi_{ij} = 1 \) if \( \pi(i) = j \) and \( \Pi_{ij} = 0 \), otherwise. The convex hull of all permutation matrices is called the Birkhoff polytope and represented as \( \{ X \in [0,1]^{n \times n} | \sum_{i \in [n]} X_{ij} = 1 \text{ for } j \in [n] \text{ and } \sum_{j \in [n]} X_{ij} = 1 \text{ for } i \in [n] \} \).
Similarly, a permutation \( \pi \) associated with a permutation \( \pi \) is defined as \( \pi = (\pi(1), \ldots, \pi(n)) \).

The convex hull of all permutation vectors is called the permutahedron.

d) Spanning Trees: Given an undirected graph \( G \), a spanning tree of the graph \( G \) is a subgraph without cycles that contains all vertices in \( G \).

B. Hedge Algorithm in the Experts Setting

Consider the simple case where \( C = \{ e_1, \ldots, e_n \} \), where \( e_i \in \{0,1\}^n \) (i = 1, . . . , n) is the unit vector whose the \( i \)-th component is 1 and other components are 0s. This case is called the experts setting. In the experts setting, there are \( n \) experts and at each trial, the player chooses one of the experts and follows its prediction. This setting naturally generalizes the case where the player has several fixed choices. The Hedge algorithm was developed by Freund and Schapire [12].

In the experts setting, given a parameter \( \eta > 0 \) and the initial distribution \( p_1 \in \Delta_n \) over experts, at each trial \( t \), the Hedge algorithm chooses \( c_t = e_{i_t} \), randomly according to the distribution \( p_t \) s.t.

\[
p_{t,i} = e^{-\eta \sum_{r=1}^{t-1} \tau_{r,i}/Z_{t-1}} \quad \text{for } i = 1, \ldots, n,
\]

where \( Z_{t-1} = \sum_{i=1}^n e^{-\eta \sum_{r=1}^{t-1} \tau_{r,i}} \).

**Theorem 1** (Freund & Schapire [12]). The regret of Hedge algorithm is \( O(\sqrt{T \ln n}) \) and \( O(\sqrt{L^* \ln n}) \) with \( \eta = \sqrt{\ln n/T} \) and \( \eta = \ln(1 + \sqrt{2 \ln n})/\sqrt{L^*} \), respectively, where \( L^* = \min_{c \in [n]} \sum_{i=1}^n \ell_t,i \).

Note that the regret bound \( O(\sqrt{L^* \ln n}) \) has a matching lower bound and thus the bound is optimal (for proofs, see, e.g., [7], [20], [34]). Prior knowledge, including Annoying issue is that, to obtain the optimal regret bound, we need to know the horizon \( T \) and the cumulative loss \( L^* \) of the best expert a priori. There are several variants of the Hedge, including the work of Koolen and Van Erven [25], which adaptively tune the time-dependent parameter \( \eta \) instead of the fixed \( \eta \) and attain the same optimal bound in the worst case and better bounds in some easy cases.

III. SIMULATION OF HEDGE OVER LARGE DECISION SPACE

The first approach to deal with large decision spaces is to simulate Hedge efficiently. The computational bottleneck of Hedge is to sample decisions \( c_t \in C \), where \( C \) is possibly exponentially large, according to the exponential weights \( p_t \).

\[
p_{t,c} \propto e^{-\eta \sum_{r=1}^{t-1} \tau_{r,c}/p_r}.
\]

There is no known general efficient method to simulate the sampling and it would be computationally hard for certain classes of \( C \). This is because such algorithms implies randomized offline linear optimization algorithms over \( C \) (possibly NP-hard) via the standard online-to-batch conversion technique.

For some particular classes of \( C \), there exists efficient sampling algorithms.

a) Paths and k-Sets: For the set of paths, Takimoto and Warmuth developed the efficient sampling technique [39] using the weight pushing algorithm of Mohri [31]. The computation time of the method is \( O(m + n) \). For \( k \)-sets, the set of \( \{0,1\} \)-vectors with \( k \) ones are encoded as paths in a DAG of size \( O(nk) \) and the sampling problem over \( k \)-sets is reduced to that of paths in a DAG with \( O(nk) \) nodes and edges.

b) Permutation Matrices and Vectors: For perfect matchings (permutation matrices), Cesa-Bianch and Lugosi pointed out that computing weights of all permutation matrices can be reduced to computing the permanent [8] and thus can be (approximately) computed by the polynomial time approximating algorithms of Jerrum et al. [21], which takes \( O(n^{10}) \) time. For permutation vectors, Ailon proposed an improved sampling scheme whose running time is \( O(n \log n) \) [2].

c) Spanning Trees: For spanning trees, to the best of our knowledge, there is no known polynomial time algorithm (in terms of \( m \) and \( n \)). Cesa-Bianchi and Lugosi also mentioned that the Markov chain-based method of Propp and Wilson [35] could be used to sample spanning trees according to exponential weights efficiently [8].

Other examples of combinatorial objects include initial segments of lists, for which Warmuth et al. proposed an efficient sampling scheme [41].

For the bandit cases, Cesa-Bianchi and Lugosi proposed COMBAND, a variant of Hedge using the sampling scheme over \( C \). The COMBAND employs the sampling scheme over the decision set \( C \) w.r.t. exponential weights and thus can be applied to combinatorial objects listed above. For paths in the bandit cases, Awerbuch and Kleinberg developed a technique using the barycentric spanner and obtained non-trivial regret bounds [5] (similar bounds are obtained by Mcmahan and Blum [28]). György et al. considered paths in the semi-bandit cases [18]. For the bandit setting, Dani et al. developed a method with \( O(\sqrt{T}) \) regret bounds using the barycentric spanner and the sampling scheme [10]. For permutation vectors, Ailon et al. proposed a generalization of Ailon’s sampling scheme for the bandit setting [3].

As a summary, an advantage of this approach is that it is often easier to analyze the regret in the full-information, semi-bandit and bandit settings, once we assume that there is an efficient sampling algorithm according to the rule (2). However, the sampling scheme needs careful design for specific classes.

IV. ONLINE MIRROR DESCENT/FTRL WITH PROJECTION AND DECOMPOSITION

The second approach for combinatorial online prediction is to relax the problems from discrete to continuous convex ones.
More precisely, the strategy is as follows:

1) (Convex relaxation) Relax the decision set \( \mathcal{C} \) to \( \text{Relax}(\mathcal{C}) \) (which includes \( \text{conv}(\mathcal{C}) \), typically \( \text{Relax}(\mathcal{C}) = \text{conv}(\mathcal{C}) \)) and use an online linear optimization algorithm over \( \text{Relax}(\mathcal{C}) \).

2) (Rounding, or Decomposition) "Round" predictions \( x_t \in \text{Relax}(\mathcal{C}) \) of the online linear optimization algorithm over \( \text{Relax}(\mathcal{C}) \) to discrete ones \( c_t \in \mathcal{C} \).

More precisely, rounding is defined as, given \( x \in \text{Relax}(\mathcal{C}) \), to choose \( c \in \mathcal{C} \) randomly so that \( E[c] \leq \alpha x \) for some \( \alpha \geq 1 \), where the expectation is defined in terms of the algorithm’s randomness. In particular, the decomposition, which is a special case of rounding, is to find a convex combination of \( c \in \mathcal{C} \) satisfying \( x = \sum_{c \in \mathcal{C}} \alpha_c c \) with \( \alpha \in [0, 1]^{\mathcal{C}} \) and \( \sum_{c \in \mathcal{C}} \alpha_c = 1 \). A decomposition method implies a rounding method with \( E[c] = x \) with \( \alpha = 1 \). A detailed description of the scheme is shown in Algorithm 1. With these components, it is easy to construct an combinatorial online prediction algorithm with regret \( O(\sqrt{T}) \) in the full-information setting.

**Theorem 2.** Suppose that an online linear optimization algorithm \( A \) with a decision set \( \text{Relax}(\mathcal{C}) \) has a regret bound \( O(\sqrt{T}) \). Then, Algorithm 1 has \( \alpha \)-regret \( O(\sqrt{T}) \).

**Proof.**

\[
(\alpha\text{-Regret}) = \sum_{t=1}^{T} E[c_t, \ell_t] - \alpha \min_{c^* \in \mathcal{C}} \sum_{t=1}^{T} c^* \cdot \ell_t \\
\leq \alpha \sum_{t=1}^{T} x_t \cdot \ell_t - \alpha \min_{c^* \in \text{Relax}(\mathcal{C})} \sum_{t=1}^{T} c^* \cdot \ell_t \\
(\text{by definition of rounding and } \mathcal{C} \subset \text{Relax}(\mathcal{C})) \\
= O(\alpha \sqrt{T}) \quad (\text{by definition of } A).
\]

\[\square\]

**A. FTRL / OMD and Projection**

For constructing an online linear optimization algorithm, we can use the **Follow the Regularized Leader (FTRL)** or the **Online Mirror Descent (OMD)** over \( \text{Relax}(\mathcal{C}) \). Let \( R : \mathcal{X} \to \mathbb{R} \) (where \( \mathcal{X} \subset \mathbb{R}^n \)) be a twice-differentiable strictly convex function. The FTRL over \( \text{Relax}(\mathcal{C}) \) with the regularizer \( R \) is defined as, for \( t = 1, \ldots, T \),

\[
x_t = \arg \max_{x \in \text{Relax}(\mathcal{C})} \sum_{\tau=1}^{t-1} x \cdot \ell_{\tau} + R(x).
\]

The equivalent algorithm with the FTRL is the OMD (sometimes called lazy version of OMD) is defined as

\[
y_t = r^{-1}(r(y_{t-1}) - \eta \ell_{t-1}) \\
x_t = \arg \min_{x \in \text{Relax}(\mathcal{C})} B_R(x, y_t), \quad (\text{projection})
\]

where \( r(x) = \nabla R(x) \), \( B_R \) is the Bregman divergence associated with \( R \) and defined as \( B_R(x, y) = R(x) - R(y) - \nabla R(y) \cdot (x - y) \), and \( y_0 = 0 \). Note that \( r^{-1} \) exists since \( R \) is strictly convex. In particular, the second step of the OMD is called the **projection** step (onto \( \text{Relax}(\mathcal{C}) \)). One of computational bottleneck in this approach lies in this step.

The OMD contains the Online Gradient Descent (OGD) [44] and the Hedge as special cases. It is known that the OMD achieves regret \( O(\sqrt{T}) \) (neglecting other factors) for online linear optimization. See, e.g., Hazan’s textbook [19] for the details of the FTRL and OMD.

**B. Projection and Rounding (Decomposition)**

- **a) k-set:** Warmuth and Kuzmin developed projection and decomposition methods onto the convex hull of the \( k \)-sets in order to solve the online optimization of PCAs [42]. The convex hull can be represented as \( \{ p \in \Delta_n \mid p_i \leq 1/k, i \in [n] \} \). The convex hull of the hull w.r.t. the relative entropy can be done in linear time. The decomposition can be done in \( O(n^2) \) time.

- **b) Permutations:** Helmhold and Warmuth proposed the PermELearn for the class \( \mathcal{C} \) of permutation matrices [20]. The relaxed decision space is \( \text{Relaxed}(\mathcal{C}) = \text{conv}(\mathcal{C}) \), which is the Birkhoff polytope. The Birkhoff polytope can be represented by \( O(n) \) linear constraints. The OMD corresponds to a matrix version of Hedge and the Bregman projection is defined w.r.t. the von Neuman matrix divergence and can be approximately solved by the Sinkhorn balancing algorithm in time \( O(n^6) \). The decomposition step can be solved by iteratively finding a perfect matching in a bipartite graph in time \( O(n^4) \).

For permutation vectors, Yasutake et al. developed a different algorithm whose running time is \( O(n^2) \) per trial [43] (the running time is improved to \( O(n \log n) \) later [27]). The convex hull of the decision space is the permutahedron, described with \( O(2^n) \) linear constraints. Apparently, the projection seems hard, but only \( O(n) \) linear constraints are shown to be relevant. A simple projection algorithm w.r.t. the relative entropy can be done in \( O(n^2) \) time. The decomposition step is solved in \( O(n \log n) \) time. Lim and Wright proposed improved projection algorithms for a large class of Bregman divergences whose running time is \( O(n \log n) \) as well [27].
c) Spanning trees: Koolen et al. proposed the Component Hedge which generalizes the framework of Helmbold and Warmuth [20] for wider classes of combinatorial objects including spanning trees [26]. The convex hull of the spanning trees is represented with $O(n^3)$ variables and linear constraints and the unnormalized relative entropy is the instance of the Bregman divergence. The decomposition step can be done by iteratively solving linear optimization of the spanning trees at most $n$ times.

d) Paths: Koolen et al. also considered projection and decomposition steps for paths [26]. For paths in a DAG with the source and sink nodes, the convex hull of them corresponds to the flow polytope, which is described with $O(n+m)$ linear constraints. So, the Bregman projection onto the flow polytope is a convex optimization over $O(n+m)$ constraints. Rounding can be done in $O(n+m)$ time.

Suehiro et al. provided a unified framework of the projection and the decomposition steps for several classes of combinatorial objects characterized with submodular functions, which includes $k$-sets, permutation vectors, spanning trees and so on [38].

C. Bandit Settings

Uchiya et al. and Kale et al. independently considered a decomposition method for $k$-sets in the semi-bandit setting [24], [40]. Combes proposed a general framework for projection and decomposition steps for the bandit setting [9].

V. OFFLINE-TO-ONLINE CONVERSION METHODS

The third approach to online combinatorial prediction is to “convert” offline linear optimization algorithms over $C$ to online ones. One of advantages of this approach is that, there are many offline linear optimization algorithms for many classes $C$ and they are often efficient.

A. Offline Exact Optimization Algorithms to Online

For 1-regret minimization, we assume an offline linear optimization algorithm $OL$ over $C$. That is, given a loss vector $\ell \in [0, 1]^n$, $OL$ outputs $\arg \min_{c \in C} c^* \cdot \ell$. A naive way to use the offline algorithm is to predict the best offline minimizer so far, given past loss functions $\ell_1, \ldots, \ell_{t-1}$. The strategy is called the Follow the Leader (FTL). Unfortunately, the regret of FTL is $\Omega(T)$ in general (see, e.g., Shalev-Shwartz’s survey for the proof [37]).

The Follow the Perturbed Leader (FPL), a slight modification of FTL achieves the regret $O(\sqrt{T})$ [23]. At each trial $t$, the FPL chooses $c_t = \arg \min_{c \in C} c \cdot (\eta \sum_{r=1}^{t-1} \ell_r + Z_t)$, where $Z_t$ is a random vector drawn according to some distribution (e.g., uniform distribution over $[-1, 1]^n$ or the component-wise exponential distributions). There are some variants of the FPL with better bounds in some case (see, Devroye et al. [11], Neu and Bartok [33] and Abernethy et al. [1]). For the semi-bandit setting, Neu developed an efficient variant of the FPL [32].

Since we only need an offline linear optimization algorithm for $C$, many classes of combinatorial concepts can be dealt with the FPL, including all examples raised in previous sections. However, a drawback of the FPL is that the regret bounds are sometimes inferior to bounds of other types. But recent analyses improving the gaps in full information, semi-bandit cases [33]. Also, as far as we know, there is no non-trivial extension of the FPL for the bandit setting.

B. Offline Approximation Algorithms to Online

For many classes of $C$, exact linear optimization of $C$ is NP-hard and thus it is natural to assume that there is only an approximation algorithm for $C$ available. More precisely, an $\alpha$-approximation algorithm for $C$ ($\alpha \geq 1$), given a loss vector $\ell \in [0, 1]^n$ as input, outputs $c \s.t. c \cdot \ell \leq \alpha \min_{c^* \in C} c^* \cdot \ell$. The FPL does not work well with approximation algorithms in general. Its regret bound becomes $O(\alpha^T \sqrt{T})$ [23].

Kakade et al. developed a better online algorithm using an $\alpha$-approximation algorithm for $C$, which achieves $O(\alpha^{T/2})$ regret bounds [22]. Their method basically follows the second approach in the previous section. Roughly speaking, their method is a combination of the OGD over $\text{conv}(C)$ with projection and decomposition steps. The key technique is to perform (approximate) projection and decomposition steps based on the Frank-Wolf method which iteratively solves linear optimization problems over $\text{conv}(C)$. To implement the Frank-Wolf method, the $\alpha$-approximation algorithm is used as a key component. The Kakade et al.’s method, however, takes $O(T)$ computation time per trial and thus takes $O(T^2)$ time after $T$ trials. Garber proposed an improved conversion algorithm with running time $O(\sqrt{T} \log T)$ per trial, while preserving the same regret bounds [17].

Fujita et al. considered a mild but natural assumption for $C$ [14]. Their assumption is that there exists a relaxation-based approximation algorithm for $C$. That is, there exists an algorithm $A$ such that, for some closed convex space $\text{Relax}(C)$ which includes $C$, given a loss vector $\ell \in [0, 1]^n$, $A$ outputs $c \in \text{Relax}(C)$ satisfying $c \cdot \ell \leq \alpha \min_{c^* \in \text{Relax}(C)} c^* \cdot \ell$. Under this assumption, their algorithm, again, a combination of the OGD and projection and decomposition steps achieves $(\alpha(\varepsilon^2) \sqrt{T})$ regret bounds with $O((1/\varepsilon^2))$ time complexity per trial.

VI. CONCLUSION

In this paper, we surveyed recent results in online combinatorial prediction in the adversarial setting. There are many open problems. For example, more efficient algorithms for particular classes are needed. Better regret bounds of projection-decomposition based methods and the FPL-based methods for the bandit setting are still open.

Another interesting direction is to application of combinatorial online prediction techniques to other areas, such as online scheduling [16], boosting over compressed data [15]. In particular, for practical applications, data structures BDD [4], MDD [29] and ZDD [30] can be useful. These data structures succinctly represent combinatorial sets as Dags in which each combinatorial object corresponds to a path. Therefore, by combining these structures, the combinatorial online prediction problem over any combinatorial decision set is reduced to that of paths. For example, Sakaue et al. considered a combination
ACKNOWLEDGMENTS

This work is supported in part by JSPS KAKENHI Grant Number JP16K00305.

REFERENCES


