Playing Games With Approximation Algorithms

Sham M. Kakade
University of Pennsylvania

Adam T. Kalai

Katrina Ligett
Carnegie Mellon University

Follow this and additional works at: http://repository.upenn.edu/statistics_papers
Part of the Statistics and Probability Commons, and the Theory and Algorithms Commons

Recommended Citation

At the time of publication, author Sham M. Kakade was affiliated with Toyota Technological Institute, Chicago. Currently (August 2016), he is a faculty member at the Statistics Department at the University of Pennsylvania.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/statistics_papers/117
For more information, please contact repository@pobox.upenn.edu.
Playing Games With Approximation Algorithms

Abstract
In an online linear optimization problem, on each period $t$, an online algorithm chooses $s_t \in S$ from a fixed (possibly infinite) set $S$ of feasible decisions. Nature (who may be adversarial) chooses a weight vector $w_t \in \mathbb{R}^n$, and the algorithm incurs cost $c(s_t, w_t)$, where $c$ is a fixed cost function that is linear in the weight vector. In the full-information setting, the vector $w_t$ is then revealed to the algorithm, and in the bandit setting, only the cost experienced, $c(s_t, w_t)$, is revealed. The goal of the online algorithm is to perform nearly as well as the best fixed $s \in S$ in hindsight. Many repeated decision-making problems with weights fit naturally into this framework, such as online shortest-path, online TSP, online clustering, and online weighted set cover.

Previously, it was shown how to convert any efficient exact offline optimization algorithm for such a problem into an efficient online bandit algorithm in both the full-information and the bandit settings, with average cost nearly as good as that of the best fixed $s \in S$ in hindsight. However, in the case where the offline algorithm is an approximation algorithm with ratio $\alpha > 1$, the previous approach only worked for special types of approximation algorithms.

We show how to convert any offline approximation algorithm for a linear optimization problem into a corresponding online approximation algorithm, with a polynomial blowup in runtime. If the offline algorithm has an $\alpha$-approximation guarantee, then the expected cost of the online algorithm on any sequence is not much larger than $\alpha$ times that of the best $s \in S$, where the best is chosen with the benefit of hindsight. Our main innovation is combining Zinkevich's algorithm for convex optimization with a geometric transformation that can be applied to any approximation algorithm. Standard techniques generalize the above result to the bandit setting, except that a "Barycentric Spanner" for the problem is also (provably) necessary as input.

Our algorithm can also be viewed as a method for playing large repeated games, where one can only compute approximate best-responses, rather than best-responses.

Keywords
approximation algorithms, regret minimization, online linear optimization

Disciplines
Statistics and Probability | Theory and Algorithms

Comments
At the time of publication, author Sham M. Kakade was affiliated with Toyota Technological Institute, Chicago. Currently (August 2016), he is a faculty member at the Statistics Department at the University of Pennsylvania.
Playing Games with Approximation Algorithms

Sham M. Kakade
Toyota Technological Institute
sham@tti-c.org

Adam Tauman Kalai*
Georgia Tech
atk@cc.gatech.edu

Katrina Ligett†
Carnegie Mellon
katrina@cs.cmu.edu

ABSTRACT

In an online linear optimization problem, on each period $t$, an online algorithm chooses $s_t \in S$ from a fixed (possibly infinite) set $S$ of feasible decisions. Nature (who may be adversarial) chooses a weight vector $w_t \in \mathbb{R}^n$, and the algorithm incurs cost $c(s_t, w_t)$, where $c$ is a fixed cost function that is linear in the weight vector. In the full-information setting, the vector $w_t$ is then revealed to the algorithm, and in the bandit setting, only the cost experienced, $c(s_t, w_t)$, is revealed. The goal of the online algorithm is to perform nearly as well as the best fixed $s \in S$ in hindsight. Many repeated decision-making problems with weights fit naturally into this framework, such as online shortest-path, online TSP, online clustering, and online weighted set cover.

Previously, it was shown how to convert any efficient exact offline optimization algorithm for such a problem into an efficient online bandit algorithm in both the full-information and the bandit settings, with average cost nearly as good as that of the best fixed $s \in S$ in hindsight. However, in the case where the offline algorithm is an approximation algorithm with ratio $\alpha > 1$, the previous approach only worked for special types of approximation algorithms.

We show how to convert any offline approximation algorithm for a linear optimization problem into a corresponding online approximation algorithm, with a polynomial blowup in runtime. If the offline algorithm has an $\alpha$-approximation guarantee, then the expected cost of the online algorithm on any sequence is not much larger than $\alpha$ times that of the best $s \in S$, where the best is chosen with the benefit of hindsight. Our main innovation is combining Zinkevich’s algorithm for convex optimization with a geometric transformation that can be applied to any approximation algorithm. Standard techniques generalize the above result to the bandit setting, except that a “Barycentric Spanner” for the problem is also (provably) necessary as input.

Our algorithm can also be viewed as a method for playing large repeated games, where one can only compute approximate best-responses, rather than best-responses.

Categories and Subject Descriptors

F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Theory

Keywords

Approximation algorithms, regret minimization, online linear optimization

1. INTRODUCTION

In the 1950’s, Hannan gave an algorithm for playing repeated two-player games against an arbitrary opponent [7]. His was one of the earliest algorithms with the no-regret property: against any opponent, his algorithm achieved expected performance asymptotically near that of the best single action, where the best is chosen with the benefit of hindsight. Put another way, after sufficiently many rounds, someone using his algorithm would not benefit (significantly) by being able to change his actions to any single action, even if this action could be chosen after observing the opponent’s play. Kalai and Vempala showed that Hannan’s approach can be used to efficiently solve online linear optimization problems as well [8]. Hannan’s algorithm relied on the ability to find best responses to an opponent’s play history. Informally speaking, Kalai and Vempala replaced this best-reply computation with an efficient black-box optimization algorithm (the number of calls to that algorithm on a sequence of length $T$ was $O(\sqrt{T})$ [8]). However, the above approach breaks down when one can only approximately solve the offline optimization problem efficiently or one can only compute approximate best responses. That is the focus of the present paper.

In an offline optimization problem, one must select a single decision $s$ from a known set of decisions $S$, in order to minimize a known cost function. In an online linear optimization problem, a weight vector $w \in \mathbb{R}^n$ is given as input, and the cost function $c(s, w)$ is assumed to be linear in $w$. Many combinatorial optimization problems fit into this framework, including traveling salesman problems (where $S$ consists of a subset of paths in a graph), clustering ($S$ is partitions of a graph), weighted set cover ($S$ is the set of
covers), and knapsack ($S$ is the set of feasible sets of items and weights correspond to item valuations).

Each of these problems has an online sequential version, in which on every period the player must select her decision without knowing that period’s cost function. That is, there is an unknown sequence of weight vectors $w_1, w_2, \ldots \in \mathbb{R}^n$ and for each $t = 1, 2, \ldots$ the player must select $s_t \in S$ and pay $c(s_t, w_t)$. In the full-information version, the player is then informed of $w_t$, while in the bandit version she is only informed of the value $c(s_t, w_t)$. (The name bandit refers to the similarity to the classic multi-armed bandit problem [10].)

The player’s goal is to achieve low average cost. In particular, we compare her cost with that of the best fixed decision: she would like her average cost to approach that of the best single point in $S$, where the best is chosen with the benefit of hindsight. This difference, $\frac{1}{T} \sum_{t=1}^{T} c(s_t, w_t) - \min_{s \in S} \frac{1}{T} \sum_{t=1}^{T} c(s, w_t)$, is termed regret.

Prior work showed how to convert an exact algorithm for the offline problem into an online algorithm with low regret, both in the full-information setting and in the bandit setting. In particular, Kalai and Vempala showed [8] that using Hannan’s approach [7], one can guarantee $O(T^{1/2})$ regret for any linear optimization problem, in the full-information version, as the number of periods $T$ increases. It was later shown [1, 9, 5] how to convert exact algorithms to achieve $O(T^{1/3})$ regret in the more difficult bandit setting.

This prior work was actually a reduction showing that one can solve the online problem nearly as efficiently as one can solve the offline problem. (They used the offline optimizer as a black box.) However, in many cases of interest, such as online combinatorial auction problems [2], even the offline problem is NP-hard. Hannan’s “follow-the-perturbed-leader” approach can also be applied to some special types of approximation algorithms, but fails to work directly in general. Finding a reduction that maintains good asymptotic performance using general approximation algorithms was posed as an open problem [8]; we resolve this problem.

In this paper, we show how to convert any approximation algorithm for a linear optimization problem into an algorithm for the online sequential version of the problem, both in the full-information setting and in the bandit setting. Our reduction maintains the asymptotic approximation guarantee of the original algorithm, relative to the average performance of the best static decision in hindsight. Our new approach is inspired by Zinkevich’s algorithm for the problem of minimizing convex functions over a convex feasible set $S \subseteq \mathbb{R}^n$ [11]. However, the application is not direct and requires a geometric transformation that can be applied to any approximation algorithm.

**Example 1 (Online metric TSP).** Every day, a delivery company serves the same $n$ customers. The company must schedule its daily route without foreknowledge of the traffic on each street. The time on any street may vary unpredictably from day to day due to traffic, construction, accidents, or even competing delivery companies. In online metric TSP, we are given a undirected graph $G$, and on every period $t$, we must output a tour that starts at a specified vertex, visits all the vertices at least once, then returns to the initial vertex. After we announce our tour, the traffic patterns are revealed (in the full-information setting, the costs on all the edges; in the bandit setting, just the cost of the tour) and we pay the cost of the tour.

**Example 2 (Online weighted set cover).** Every financial quarter, our company hires vendors from a fixed pool of subcontractors to cover a fixed set of tasks. Each subcontractor can handle a known, fixed subset of the tasks, but their price is only announced at the end of the quarter and varies from quarter to quarter. In online weighted set cover, the vendors are fixed sets $P_1, \ldots, P_n \subseteq [m]$. Each period, we choose a legal cover $s_t \subseteq [n]$, that is, $\bigcup_{i \in s_t} P_i = [m]$. There is an unknown sequence of cost vectors $w_1, w_2, \ldots \in [0, 1]^n$, indicating the quarterly vendor costs. Each quarter, our total cost $\sum_{i \in s_t} c(s_t, w_t)$ is the sum of the costs of the vendors we chose for that quarter. In the full-information setting, at the end of the quarter we find out the price charged by each of the subcontractors; in the bandit setting, we receive a combined bill showing only our total cost.

### 1.1 Hannan’s approach

In this section, we briefly describe the previous approach [8] for the case of exact optimization algorithms based on Hannan’s idea of adding perturbations. We begin with the obvious “follow-the-leader” algorithm which, at each period, picks the decision that is best against the total (equivalently, average) of the previous weight vectors. This means, on period $t$, choosing $s_t = A(\sum_{r=1}^{t-1} w_r)$, where $A$ is an algorithm that, given a cost vector $w$, produces the best $s \in S$. Hannan’s perturbation idea, in our context, suggests using $s_t = A(p_t + \sum_{r=1}^{t-1} w_r)$ for uniformly random perturbation $p_t \in [0, \sqrt{T}]^n$. One can bound the expected regret of following-the-perturbed-leader to be $O(T^{-1/2})$, disregarding other parameters of the problem.

Kalai and Vempala [8] note that Hannan’s approach maintains an asymptotic $\alpha$-approximation guarantee when used with $\alpha$-approximation algorithms with a special property they call $\alpha$-point-wise approximation, meaning that on any input, the solution they find differs from the optimal solution by a factor of at most $\alpha$ in every coordinate. They observe that a number of algorithms, such as the Goemans-Williamson max-cut algorithm [6], have this property. Baldcan and Blum [2] observe that the previous approach applies to another type of approximation algorithm: one that uses an optimal decision for another linear optimization problem, for example, using MST for TSP. It is also not difficult to see that a FPTAS can be used to get a $(1+\epsilon)$-competitive online algorithm. We further note that the Hannan-Kalai-Vempala approach extends to approximation algorithms that perform a simple type of randomized rounding where the randomness does not depend on the input.

In Appendix A, we use an explicit example based on the greedy set-cover approximation algorithm to illustrate how Hannan’s approach fails on more general approximation algorithms.

### 1.2 Informal statement of results

The main result of this paper is a general conversion from any approximate linear optimization algorithm to an approximate online version in the full-information setting (Sec-

---

1This approach fails even on a two-decision problem, where the costs of the two decisions are $(0,0,0)$ during the first period and then alternate $(1,0), (0,1), (1,0), \ldots$, thereafter.
The extension to the bandit setting (Section 4) uses well-understood techniques, modulo one new issue that arises in the case of approximation algorithms. We summarize the problem, our approach, and our results here.

We assume there is a known compact convex set $W \subseteq \mathbb{R}^n$ of legal weight vectors (in many cases $W = [0, 1]^n$), and a cost function $c : S \times W \to [0, 1]$ that is linear in its second argument, that is, $c(s, av + bw) = ac(s, v) + bc(s, v)$ for all $s \in S$, $a, b \in \mathbb{R}$, and $v, w, av + bw \in W$. The generalization to $[0, M]$-bounded cost functions for $M > 0$ is straightforward. We assume that we have a black-box $\alpha$-approximation algorithm, which we abstract as an oracle $A$ such that, for all $w \in W$, $c(A(w), w) \leq \alpha \min_{s \in S} c(s, w)$. That is, do not assume that our approximation oracle can optimize in every direction. In the full-information setting, we assume our only access to $S$ is via the approximation algorithm; in the bandit setting, we need an additional assumption, which we describe below.

For simplicity, in this paper, we focus on the non-adaptive setting, in which the adversary’s choices of $w_i$ can be arbitrary but must be chosen in advance. In the adaptive setting, on period $t$, the adversary may choose $w_t$ based on $s_1, w_1, \ldots, s_{t-1}, w_{t-1}$. There is no clear reason why the results presented here cannot be extended to the adaptive setting.

For $\alpha$-approximation algorithms, it is natural to consider the following notion of $\alpha$-regret, in both the full-information and the bandit-settings. It is the difference between the algorithm’s average cost and $\alpha$ times the cost of the best $s \in S$, that is, $\frac{1}{T} \sum_{t=1}^{T} c(s, w_t) - \alpha \min_{s \in S} \frac{1}{T} \sum_{t=1}^{T} c(s, w_t)$.

### 1.2.1 Full-information results

Our approach to the full-information problem is inspired by Zinkevich’s algorithm (for a somewhat different problem)[11], which uses an exact projection oracle to create an online algorithm with low regret. An exact projection oracle $\Pi$ is an algorithm which can produce $\arg\min_{x \in J} \|x - y\|$ for all $y \in \mathbb{R}^n$, where $J$ is the “feasible region” (in Zinkevich’s setting, a compact convex subset of $\mathbb{R}^n$). The main algorithm presented in Zinkevich’s paper, Greedy Projection, determines its decision $x_t$ at time $t$ as $x_t = \Pi(y_{t-1} - \eta w_{t-1})$, where $\eta$ is a parameter called the learning rate and $w_{t-1}$ is the cost vector at time ($t - 1$). One can view the approach in this paper as providing a method to simulate a type of “approximate” projection oracle using an approximation algorithm. In Section 3 we show the following:

**Result 1.1.** Given any $\alpha$-approximation oracle to an offline linear-optimization problem and any $T, T_0 \geq 1$, $w_1, w_2, \ldots \in W$, our (full-information) algorithm outputs $s_1, s_2, \ldots \in S$ achieving

$$E \left[ \frac{1}{T} \sum_{t=T_0+1}^{T+T} c(s_t, w_t) \right] - \alpha \min_{s \in S} \frac{1}{T} \sum_{t=T_0+1}^{T+T} c(s, w_t) = O(\alpha n / \sqrt{T}).$$

The algorithm makes $\text{poly}(n, T)$ calls to the approximation oracle.

Note that the above bound on expected $\alpha$-regret holds simultaneously for every window of $T$ consecutive periods ($T$ must be known by the algorithm). We easily inherit this useful adaptation property of Zinkevich’s algorithm. It is not

### 1.2.2 Bandit results

Previous work in the bandit setting constructs an “exploration basis” to allow the algorithm to discover better decisions [1, 9, 5]. In particular, Awerbuch and Kleinberg [1] introduce a so-called Barycentric Spanner (BS) as their exploration basis and show how to construct one from an optimization oracle $A : \mathbb{R}^n \to S$. However, in the case where the oracle (exact or approximate) only accepts inputs in, say, the positive orthant, it may be impossible to extract an exploration basis. Hence, we assume that we are given a $\beta$-BS ($\beta \geq 1$ is an approximation factor for the BS) for the problem at hand as part of the input. Note that the $\beta$-BS only needs to be computed once for a particular problem and then can be reused for all future instances of that problem.

Given a $\beta$-BS, the standard reduction from the bandit setting to the full-information setting gives:

**Result 1.2.** For any $\beta$-BS and any $\alpha$-approximation oracle to an offline linear-optimization problem and any $T, T_0 \geq 1$, $w_1, w_2, \ldots \in W$, the (bandit) algorithm in Figure 4 outputs $s_1, s_2, \ldots \in S$ achieving

$$E \left[ \frac{1}{T} \sum_{t=T_0+1}^{T+T} c(s_t, w_t) \right] \leq \alpha \min_{s \in S} \frac{1}{T} \sum_{t=T_0+1}^{T+T} c(s, w_t) + O(n\alpha^{2/3} \beta^{1/3}) / \sqrt{T}.$$

The algorithm makes $\text{poly}(n, T)$ calls to the approximation oracle.

We also show, in Section 4.1, that the assumption of a BS is necessary.

**Result 1.3.** There is no polynomial-time black-box reduction from an $\alpha$-approximation algorithm for a general linear optimization problem (without additional input) to a bandit algorithm guaranteeing low $\alpha$-regret.

### 2. Formal Definitions

We formalize the natural notion of an $n$-dimensional linear optimization problem.

**Definition 1.** An $n$-dimensional linear optimization problem consists of a convex compact set of feasible weight vectors $W \subseteq \mathbb{R}^n$, a set of feasible decisions $S$, and a cost function $c : S \times W \to [0, 1]$ that is linear in its second argument.

Due to the linearity of $c$, there must exist a mapping $\Phi : S \to \mathbb{R}^n$ such that $c(s, w) = \Phi(s) \cdot w$ for all $s \in S, w \in W$. In the case where the standard basis is contained in $W$, we have

$$\Phi(s) = (c(s, (1, 0, \ldots, 0)), \ldots, c(s, (0, \ldots, 0, 1))).$$

More generally, the mapping $\Phi$ can be computed directly from $c$ by evaluating $c$ at any set of vectors whose span includes $W$. We will assume that we have access to $\Phi$ and $c$ interchangeably. Note that previous work represented the problem directly as a geometric problem in $\mathbb{R}^n$, but in our case we hope that making the mapping $\Phi$ explicit clarifies the algorithm.
An α-approximation algorithm A (α ≥ 1) for such a problem takes as input any vector w ∈ W and outputs A(w) ∈ S such that c(A(w), w) ≤ α · min_{s ∈ S} c(s, w). To ensure that the min is well-defined, we also assume Φ(S) = {Φ(s) | s ∈ S} is compact.

Define a projection oracle Π_J : R^n → J, where Π_J(x) = argmin_{z ∈ J} ||x − z|| is the unique projection of x to the closest point z in the set J.

Define W_k = {aw | a ≥ 0, w ∈ W} ⊆ R^n. Note that W_k is convex, which follows from the convexity of W. We assume that we have an exact projection oracle Π_{W_k}. This is generally straightforward to compute. In many cases, W = [0, 1]^n, in which case W_k is the positive orthant and Π_{W_k}(w)[i] is simply max(w[i], 0), where w[i] denotes the ith component of vector w. More generally, given a membership oracle to W (and a point w ∈ W and appropriate bounds on the radii of contained and containing balls), one can approximate the projection to within any desired accuracy ε > 0 in time poly(n, log(1/ε)).

We also assume, for convenience, that A : W_k → S because we know that A(w) can be chosen to be equal to A(aw) for any a > 0. Again, given a membership oracle to W one can find v ∈ V which is within ε of being a scaled version of w using time poly(n, 1/ε). However, the restriction on the approximation algorithm’s domain is important because many natural approximation algorithms only apply to restricted domains such as non-negative weight vectors.

In an online linear optimization problem, there is a sequence w_1, w_2, ..., w_T of weight vectors. Due to the linearity of the problem, an offline optimum can be computed using an exact optimizer, that is, min_{s ∈ S} Σ_{t=1}^T Φ(s) · w_t gives the average cost of the best single decision if one had to use a single decision during all time periods t = 1, 2, ..., T. Similarly, an α-approximation algorithm, when applied to Σ_{t=1}^T w_t, gives a decision whose average cost is not more than a factor α larger than that of the offline optimum.

Definition 2. In a full-information online linear optimization problem, there is an unknown sequence of weight vectors w_1, w_2, ..., w_T of weight vectors. On each period, the decision-maker chooses a decision s_t ∈ S based on s_1, w_1, s_2, w_2, ..., s_{t−1}, w_{t−1}. Then w_t is revealed and the decision-maker incurs cost c(s_t, w_t).

Finally, we define the bandit version of the problem, in which the algorithm finds out only the cost of its decision, c(s_t, w_t), but not w_t itself.

Definition 3. In a bandit online linear optimization problem, there is an unknown sequence of weight vectors w_1, w_2, ..., w_T of weight vectors. On each period, the decision-maker chooses a decision s_t ∈ S based only upon s_1, c(w_1, s_1), ..., s_{t−1}, c(w_{t−1}, s_{t−1}). Then only the cost c(s_t, w_t) is revealed.

The performance of an online algorithm is measured by comparing its cost on a sequence of weight vectors with the cost of the best static decision for that sequence.

Definition 4. The α-regret of an algorithm that selects decisions s_1, ..., s_T ∈ S is defined to be

α-regret(s_1, w_1, ..., s_T, w_T) = α · min_{s ∈ S} Σ_{t=1}^T c(s, w_t) − 1/T Σ_{t=1}^T c(s, w_t).

The term regret by itself refers to 1-regret.

For x, y ∈ R^n and W ⊆ R^n, we say x dominates y if x · w ≤ y · w for all w ∈ W (equivalently, for all w ∈ W_k). Define K ⊆ R^n to be the convex hull of Φ(S).

K = \{ \sum_{i=1}^{n+1} λ_i Φ(s_i) | s_i ∈ S, λ_i ≥ 0, \sum_i λ_i = 1 \}.

Note that min_{w ∈ K} x · w = min_{s ∈ S} c(s, w) for all w ∈ W. The cost of any point in K can be achieved by choosing a randomized combination of decisions s ∈ S. However, we must find such a combination of decisions and compute projections in our setting, where our only access to S is via an approximation oracle.

3. FULL-INFORMATION ALGORITHM

We now present our algorithm for the full-information setting. Define z_t = x_t − ηw_t. Intuitively, one might like to play z_t on period t + 1 because z_t has less cost than x_t against w_t. Unfortunately, z_t may not be feasible. In the Greedy Projection algorithm of Zinkevich, the decision played on period t + 1 is the projection of z_t into the feasible set. Our basic approach is to implement an approximate projection algorithm and play the approximate projection of z_t on step (t + 1).

There are a number of technical challenges to this approach. First, we only have access to an α-approximation oracle with which to implement this. Due to the multiplicative nature of this approximation, we proceed by attempting to project into the set αK, where αK = \{ αx | x ∈ K \}. Second, even if we could do this perfectly (which is not possible), this would still not result in a feasible decision. We then must find a way to play a feasible decision.

We can intuitively view our algorithm as follows. The algorithm keeps track of a parameter x_t, which we can think of as the attempt to project z_{t−1} into αK (though this is not done exactly, as x_t is not even in αK). We show that if the algorithm actually were allowed to play x_t then it would have low α-regret. Our algorithm uses this x_t to find a randomized feasible decision s_t. We show that the expected cost of this random feasible decision s_t is no larger than that of the infeasible x_t.

Our algorithm for the full-information setting is based on the approximate projection routine defined in Figure 3.

Algorithm 3.1. On period t, we choose an arbitrary s_t (which could be selected by running the approximation oracle on any input) and let x_t = Φ(s_t). On period t, we play x_t and let

(x_{t+1}, s_{t+1}) = \text{APPROX-PROJ}(x_t − ηw_t, s_t, x_t).

It may be helpful to the reader to note that the sequence x_{t+1} is deterministically determined (if the approximation oracle is deterministic) by the sequence of weights w_1, ..., w_{t−1}, while s_{t+1} is necessarily randomized.

Note that this definition differs from the standard definition in R^n where x dominates y if x[i] ≥ y[i] for all i but resembles the game-theoretic notion of dominant strategies.
In Section 3.1, we show that if we had a particular kind of approximate projection algorithm, then the \( x_t \) values produced by that algorithm would have (hypothetical) low \( \alpha \)-regret. In Section 3.2, we show how to extend the domain of any approximation algorithm, which allows us to construct such an approximate projection algorithm: the APPROX-PROJ algorithm used in Algorithm 3.1. We also show that the cost of the (infeasible) decision \( x_t \) it produces can only be larger than the expected cost incurred by the feasible decision \( s_t \) it also generates. This will allow us to prove our main theorem in the full-information setting:

**Theorem 3.2.** Consider an \( n \)-dimensional online linear optimization problem with feasible set \( S \) and mapping \( \Phi: S \to \mathbb{R}^n \). Let \( A \) be an \( \alpha \)-approximation algorithm and take \( R, W \geq 0 \) such that \( \|\Phi(A(w))\| \leq R \) and \( \|w\| \leq W \) for all \( w \in \mathcal{W} \). For any fixed \( w_1, w_2, \ldots, w_T \in \mathcal{W} \) and any \( T \geq 1 \), with \( \eta = \frac{(\alpha+1)R}{\sqrt{W^2T}} \), and \( \lambda = \frac{(\alpha+1)}{\sqrt{4(\alpha+2)^2T}} \). Algorithm 3.1 achieves expected \( \alpha \)-regret at most

\[
E\left[\frac{1}{T} \sum_{t=1}^{T} c(s_t, w_t)\right] - \alpha \min_{s \in S} \frac{1}{T} \sum_{t=1}^{T} c(s, w_t) \leq \frac{(\alpha + 2)RW}{\sqrt{T}}.
\]

Each period, the algorithm makes at most \( 4(\alpha + 2)^2T \) calls to \( A \) and \( \Phi \).

We present the proof of Theorem 3.2 in Section 3.3. To get Result 1.1 in the introduction, we note that it is possible to get to what bounds on \( W \) and \( R \) by a simple change of basis so that \( R W = O(n) \). It is possible to do this from the set \( W \) alone. In particular, one can compute a 2-barycentric spanner (BS) \( e_1, \ldots, e_n \) for \( \mathcal{W} \) [1] and perform a change of basis so that \( \Phi(e_1), \ldots, \Phi(e_n) \) is the standard basis (as we describe in greater detail in Section 4). By the definition of a 2-BS, this implies that \( \mathcal{W} \subseteq [-2,2]^n \) and hence \( R = 2\sqrt{n} \) is a satisfactory upper bound. Since we have assumed that all costs are in \([0,1]\) and the standard basis is in \( W \), this implies that \( \Phi(S) \subseteq [0,1]^n \) and hence \( R = \sqrt{n} \) is also a valid upper bound. The guarantees with respect to every window of \( T \) consecutive periods hold because our algorithm’s guarantees hold starting at arbitrary \((s_1, x_1)\) such that \( E[\Phi(s_1)] \) dominates \( x_t \).

### 3.1 Approximate Projection

We first define the notion of approximate projection. It is approximate in two senses: first, even if we had an exact optimization oracle (\( \alpha = 1 \)), we could not find the absolute closest point \( x \in K \) to any point \( z \in \mathbb{R}^n \). The second and more important sense in which it is approximate is that, because we only have an \( \alpha \)-approximate oracle, we cannot find the closest point in \( K \) or even in \( \alpha K = \{\alpha x \mid x \in K\} \).

Note that for a closed convex set \( J \subseteq \mathbb{R}^n \), if \( \Pi_J(z) = x \), then

\[
(x - z) \cdot x \leq \min_{y \in J} (x - z) \cdot y.
\]

This is essentially the separating hyperplane theorem (where \( x - z \) is the normal vector to the separating hyperplane). Also note that \( \Pi_J(x) = x \) if \( x \in J \).

---

**Figure 1:** An approximate projection oracle, for convex set \( J \subseteq \mathbb{R}^n \) and \( \delta = 0 \), returns a point \( \Pi_J^\delta(z) \in \mathbb{R}^n \) that is closer to any point \( y \in J \) than \( z \), that is, \( \forall y \in J \|\Pi_J^\delta(z) - y\| \leq \|z - y\| \).

Our approximate projection property, illustrated in Figure 1, relaxes the above condition. Define the set of \( \delta \)-approximate projections to be, for \( \delta \geq 0 \) and any \( z \in \mathbb{R}^n \),

\[
\Pi_J^\delta(z) = \{x \in \mathbb{R}^n \mid (x - z) \cdot x \leq \min_{y \in J} (x - z) \cdot y + \delta\}.
\]

It is important to note that we have not required an approximate projection to be in \( J \). However, note that in the case where the projection is in \( J \), and \( \delta = 0 \) is exactly the projection, that is, \( \Pi_J^0(x) \cap J = \{\Pi_J(x)\} \). While we refer to it as an approximate projection, it is also clearly related to a separation oracle. From a hyperplane separating \( x \) from \( J \), one can take the closest point on that hyperplane to \( x \) as an approximate projection. The difficulty is in finding a feasible such \( x \).

We now bound the \( \alpha \)-regret of the hypothetical algorithm which projects with \( \Pi_{\alpha K} \). The proof is essentially a straightforward extension of Zinkevich’s proof [11]. This lemma shows that indeed this hypothetical algorithm has a graceful degradation in quality.

**Lemma 3.3.** Let \( K \subseteq \mathbb{R}^n \) be a convex set such that \( \forall x \in K \|x\| \leq R \). Let \( w_1, \ldots, w_T \in \mathbb{R}^n \) be an arbitrary sequence. Then, for any initial point \( x_1 \in K \) and any sequence \( x_1, x_2, \ldots, x_T \) such that \( x_{t+1} \in \Pi_{\alpha K}^\delta(x_t - \eta w_t) \),

\[
\frac{1}{T} \sum_{t=1}^{T} x_t \cdot w_t - \alpha \min_{x \in K} \frac{1}{T} \sum_{t=1}^{T} x_t \cdot w_t \leq \frac{(\alpha + 1)^2 R^2}{2 \eta T} + \frac{\eta}{2T} \sum_{t=1}^{T} w_t^2 + \frac{\delta}{\eta}.
\]

**Proof.** Let \( x^* = \alpha \arg\min_{x \in K} \sum_{t=1}^{T} x \cdot w_t \), so \( x^* \in \alpha K \). We will bound our performance with respect to \( x^* \). Define the sequence \( x_t' \) by \( x_1' = x_1 \) and \( x_{t+1}' = x_{t+1} - \eta w_t \), so that \( x_t \in \Pi_{\alpha K}^\delta(x_t') \). We first claim that \( \|x_t - x^*\| \leq \|x_t' - x^*\| + 2 \delta \), that is, our attempt at setting \( x_t \) to be an approximate projection of \( x_t' \) onto \( \alpha K \) does not increase the distance to \( x^* \) significantly:

\[
(x_t - x^*)^2 = (x_t' - x_t + x_t - x^*)^2 \\
= (x_t' - x_t)^2 + (x_t - x^*)^2 + 2(x_t' - x_t) \cdot (x_t - x^*) \\
\geq 0 + (x_t - x^*)^2 - 2 \delta
\]
The last line follows from the definition of approximate projection and the fact that $x^* \in \alpha K$.

Hence, for any $t \geq 1$, because $x_t = x_t - \eta w_t$ we have

\[
(x_{t+1} - x^*)^2 \leq (x_t - \eta w_t - x^*)^2 + 2\delta \\
= (x_t - x^*)^2 + \eta^2 w_t^2 - 2\eta w_t \cdot (x_t - x^*) + 2\delta
\]

and thus

\[
w_t \cdot (x_t - x^*) \leq \frac{(x_t - x^*)^2 - (x_{t+1} - x^*)^2 + \eta^2 w_t^2 + 2\delta}{2\eta}.
\]

Using a telescoping sum of the above and the fact that

\[
(x_1 - x^*)^2 \leq (\|x_1\| + \|x^*\|)^2 \leq (\alpha + 1)^2 R^2,
\]

we get

\[
\sum_{t=1}^{T} x_t \cdot w_t - \alpha \min_{x \in K} \sum_{t=1}^{T} x \cdot w_t \leq \frac{(\alpha + 1)^2 R^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} w_t^2 + T \frac{\delta}{\eta}
\]

as desired. \hfill \Box

Note that if we set $\eta = 1/\sqrt{T}$, the sum of the first two terms of this bound would be $O(1/\sqrt{T})$. However, the last term, $\frac{T}{\eta}$, would be $O(\sqrt{T})$. Hence, we need to achieve an approximation quality of $\delta = O(1/T)$ in order for the $\alpha$-regret of our (infeasible) $x_t$ values to be $O(1/\sqrt{T})$.

### 3.2 Constructing the Algorithm

One simple method to (approximately) find a projection of $z$ into a convex set $J$, given an exact optimization oracle for $J$, is as follows. Start with a point in $x \in J$. Then choose the search direction $v = x - z$, and find a minimal point $x' \in J$ in the direction of $v$, that is, $x' \in J$ such that $x' \cdot v \leq \min_{y \in J} y \cdot v$. It can be seen that if $x$ is not minimal in the direction of $v$, then there must be a point on the segment joining $x'$ and $z$ that is closer to $z$ than $x$ was. Then repeat this procedure starting at $x'$. In the case where $z \in J$, this will be still be useful in representing $z$ nearly as a combination of points output by the minimization algorithm.5

Note that in our case if $v \in W_+$, then our approximation oracle is able to find a feasible $s \in S$ such that

\[
\Phi(s) \cdot v \leq \alpha \min_{s' \in S} \Phi(s') \cdot v = \min_{x \in \alpha K} x \cdot v.
\]

Loosely speaking, our oracle is able to perform minimization with respect to the set $J = \alpha K$ (or better). This is essentially how our algorithm will use the approximation oracle. However, as mentioned before, many approximation algorithms can only handle non-negative weight vectors or weight vectors from some other limited domain. Hence, we must extend the domain of the oracle when $v \notin W_+$.

**Extending the domain.** We would like to find a feasible $s \in S$ that satisfies the search condition $\Phi(s) \cdot v \leq \alpha \min_{s' \in S} \Phi(s')$ for a general $v \in \mathbb{R}^n$, but this is not possible only given an $\alpha$-approximation oracle that runs on only a subset of $\mathbb{R}^n$. Instead, we attempt to find a (potentially infeasible) $x \in \mathbb{R}^n$.

Note that representing a given feasible point as a convex combination of feasible points is similar to randomized metarounding [3]. It would be interesting to extend their approach, based on the ellipsoid algorithm, to our problem and potentially achieve a more efficient algorithm. Related but simpler issues arise in [4].

\[
\Phi(s) \cdot v \leq \alpha \min_{s' \in S} \Phi(s') \cdot v
\]

which does satisfy this search condition, and we also attempt to find an $s \in S$ which dominates $x$, meaning that for all $w \in W$, $c(s, w) \leq x \cdot w$. More precisely, we will construct the following oracle:

**Definition 5.** An extended approximation oracle $B : \mathbb{R}^n \rightarrow S \times \mathbb{R}^n$ is a function such that, for all $v \in \mathbb{R}^n$, if $B(v) = (s, x)$, then $x \cdot v \leq \alpha \min_{s' \in S} \Phi(s') \cdot v$ and $\Phi(s)$ dominates $x$.

Figure 2 depicts an extended approximation oracle. The following lemma demonstrates that one can construct an extended approximation oracle from an approximation oracle.

**Lemma 3.4.** Let $A : W_+ \rightarrow S$ be an $\alpha$-approximation oracle and suppose $\|\Phi(s')\| \leq R$ for all $s' \in S$. Then the following is an extended approximation oracle: If $v \in W_+$, then $B(v) = (A(v), \Phi(A(v)))$, else $B(v)$ is

\[
(A(\Pi_{W_+}(v)), \Phi(A(\Pi_{W_+}(v)))) + R(\alpha + 1) \frac{\Pi_{W_+}(v) - v}{\|\Pi_{W_+}(v) - v\|}.
\]

**Proof.** For the case where $v \in W_+$, by definition, $B(v) = (A(v), \Phi(A(v)))$ satisfies. Hence, assume $v \notin W_+$. Let $w = \Pi_{W_+}(v)$, $s = A(w)$, and $x = \Phi(s) + (\alpha + 1)R \frac{w - v}{\|w - v\|}$. Then we must show (a) $x \cdot v \leq \alpha \min_{s' \in S} \Phi(s') \cdot v$ and (b) $\Phi(s)$ dominates $x$.

We have assumed that $A$ is an $\alpha$-approximation oracle with domain $W_+$, and therefore it can accept input $w$. By the definition of $\alpha$-approximation, we have $w \cdot \Phi(s) \leq \alpha w \cdot \Phi(s')$ for all $s' \in S$. By the bound $R$, we also have that $-\alpha\|v - w\| R \leq \alpha(v - w) \cdot \Phi(s')$ for all $s' \in S$. Adding these
Approx-Proj \( (z, s, x) \)

1. Let \( (t, y) := B(x - z) \)
2. If \( x \cdot (x - z) \leq \delta + y \cdot (x - z) \), then return \( (x, s) \)
3. Else \( q = \begin{cases} s & \text{with probability } 1 - \lambda \\ t & \text{with probability } \lambda \end{cases} \)
4. Return \( \text{APPROX-PROJ}(z, q, \lambda y + (1 - \lambda)x) \)

**Figure 3:** An iterative algorithm for computing approximate projections.

The return condition of \( \text{APPROX-PROJ} \) states that \( x' \cdot (x' - z) \leq \delta + y \cdot (x' - z) \). Using the definition of an extended approximation oracle, we then get

\[
x' \cdot (x' - z) \leq \delta + \min_{s' \in \mathcal{S}} \Phi(s') \cdot (x' - z)
\]

as desired.

The proof of the second property proceeds by induction on the number of recursive calls made by \( \text{APPROX-PROJ} \). The base case holds trivially. Now suppose the inductive hypothesis holds \( E[\Phi(s)] \) dominates \( x \). We will show that if \( (t, y) = B(x - z) \), the resulting \( E[\Phi(\lambda y + (1 - \lambda)s)] \) dominates \( \lambda y + (1 - \lambda)x \).

We observe:

\[
x' \cdot w \leq (\lambda y + (1 - \lambda)x) \cdot w
\]

\[
\leq \lambda y \cdot w(1 - \lambda)x \cdot w
\]

\[
\geq \lambda \Phi(t) \cdot w + (1 - \lambda)x \cdot w
\]

\[
= \lambda \Phi(t) + (1 - \lambda)\Phi(s) \cdot w
\]

\[
= E[\Phi(s')] \cdot w.
\]

Thus, if \( \text{APPROX-PROJ} \) terminates, the desired conditions will hold.

### 3.3 Analysis

Our next lemma allows us to bound the number of calls Algorithm 3.1 makes to \( A \) and \( \Phi \) on each period.

**Lemma 3.6.** Suppose that \( \lambda, \delta > 0 \) and the magnitudes of all vectors output by the extended approximation oracle are \( \leq \sqrt{21} \times H \). Then \( \text{APPROX-PROJ}(z, s, x) \) terminates after at most \( \frac{\sqrt{21}^2 - 1}{2 \lambda^2} \) iterations.

**Proof.** Let \( H = \frac{\sqrt{21}}{2} \). To bound the number of recursive calls to \( \text{APPROX-PROJ} \), it suffices to show that the non-negative quantity \( ||x - z||^2 \) decreases by at least an additive \( \delta \lambda \) on each call and that \( ||x|| \) remains below \( H \) on successive calls. The latter condition holds because \( ||x||, ||y|| \leq H \) so \( ||\lambda y + (1 - \lambda)x|| \leq \lambda H + (1 - \lambda)H = H \).

Notice that if the procedure does not terminate on a particular call, then

\[
(x - y) \cdot (x - z) > \delta.
\]

This means that the decrease in \( ||x - z||^2 \) in a single recursive call is

\[
(x - z)^2 - (\lambda y + (1 - \lambda)x - z)^2
\]

\[
= (x - z)^2 - (\lambda(y - x) + (x - z))^2
\]

\[
= 2\lambda(x - y) \cdot (x - z) - \lambda^2(y - x)^2
\]

\[
> 2\delta^2 - \lambda^2(y - x)^2.
\]

Also, \( ||y - x|| \leq 2H \). Combining this with the previous observation gives

\[
(x - z)^2 - (\lambda y + (1 - \lambda)x - z)^2 \geq 2\delta - \lambda^24H^2 = \lambda\delta.
\]

Hence the total number of iterations of \( \text{APPROX-PROJ} \) on each period is at most \( ||x - z||^2 / (\lambda\delta) \).
This lemma gives us a means of choosing \( \lambda \). We are now ready to prove our main theorem about full-information online optimization.

**Proof of Theorem 3.2.** Take \( \eta = \frac{(\alpha+1)R}{W \sqrt{T}} \) and \( \delta = \frac{(\alpha+1)R^2}{2W} \). Since \( x_t = \Phi(s_t) \), by induction and Lemma 3.5, we have that \( \mathbb{E}[\Phi(s_t)] \) dominates \( x_t \) for all \( t \). Hence, it suffices to upper-bound \( \sum_{t=1}^{T} x_t \cdot w_t \). By Lemma 3.5, we have that \( x_t \in \Pi_{\alpha,k}(x_{t-1}) \) on each period, so by Lemma 3.3 we get

\[
\mathbb{E}[\alpha \text{-regret}] \leq \frac{1}{T} \left( \frac{(\alpha+1)^2R^2}{2\eta} + T \frac{\delta}{\eta} + \frac{\eta}{2} T W^2 \right).
\]

Applying our chosen values of \( \eta \) and \( \delta \), this gives an \( \alpha \)-regret bound of \( \frac{1}{T}((\alpha+1)RW \sqrt{T} + RW \sqrt{T}) = \frac{(\alpha+2)RW}{2\eta} \) as desired.

Now, as mentioned, the extended approximation oracle from Lemma 3.4 has the property that it returns vectors of magnitude at most \( H = \frac{1}{2} \sqrt{\frac{T}{\alpha}} = (\alpha+2)R \). Furthermore, it is easy to see that all vectors \( x_t \) have \( \|x_t\| \leq H \), by induction on \( t \). Then by Lemma 3.6, the total number of iterations of APPROX-PROJ period \( t \) is at most \( (2H\|x-z\|/\delta)^2 \leq (2(\alpha+2)RW/\delta)^2 = 4(\alpha+2)^2T \). \( \square \)

**4. Bandit Algorithm**

We now describe how to extend Algorithm 3.1 to the partial-information model, where the only feedback we receive is the cost we incur at each period. The algorithm we describe here requires access to an *exploration basis* \( e_1, \ldots, e_n \in S \), which is simply a set of \( n \) decisions such that \( \Phi(e_1), \ldots, \Phi(e_n) \) span \( \mathbb{R}^n \). (If no such decisions exist, one can reduce the problem to a lower-dimensional problem.) Following previous approaches, we will (probabilistically) try each of these decisions from time to time. As in the work of Dani and Hayes [5], we will assume that \( \Phi(e_i) \) is the standard \( i \)th basis vector, that is, \( e_i[1] = 1 \) and \( e_i[j] = 0 \) for \( j \neq i \). This assumption makes the algorithm cleaner to present, and is without loss of generality because we can always use \( \Phi(e_i) \) as our basis for representing \( \mathbb{R}^n \).

**Definition 6.** A set \( \{x_1, x_2, \ldots, x_m\} \subseteq S \) is a \( \beta \)-barycentric spanner (BS) for \( S \subseteq \mathbb{R}^n \) if, for every \( x \in S \), \( x \) can be written as \( x = \beta_1 x_1 + \ldots + \beta_n x_m \) for some \( \beta_1, \ldots, \beta_n \in [-\beta, \beta] \).

Note that we only need to construct a BS once for any problem, and then can re-use it for all future instances of the problem.

Awerbuch and Kleinberg [1] prove that every compact \( S \) has a 1-BS of size \( n \), and, moreover, give an algorithm for finding a size-\( n \) (1 + \( \epsilon \))-BS using \( \text{poly}(n, \log(1/\epsilon)) \) calls to an exact minimization oracle \( M : \mathbb{R}^n \rightarrow S \), where \( M(v) \in \arg\min_{s \in S} \Phi(s) - v \). Unfortunately, as we show in Section 4.1, one cannot find such a BS using a minimizer (exact or approximate) whose domain is not all of \( \mathbb{R}^n \). Moreover, we show that one cannot guarantee low regret for the bandit problem using just a black-box optimization algorithm \( A : \mathcal{W}_N \rightarrow S \).

Hence, we assume that we are given a \( \beta \)-BS for the problem at hand as part of the input. We feel that this is a reasonable assumption. For example, note that it is easy to find such a basis for TSP and set cover with \( \beta = \text{poly}(n) \): In the case of set cover, one can take the \( n \) covers consisting of all sets but one.6 In the case of TSP, we can start with any

Given \( \delta, \eta, \gamma > 0 \) and an initial point \( s_1 \) as input, set \( s_1 = \Phi(s_1) \). Perform a change of basis so that \( \Phi(e_1), \ldots, \Phi(e_n) \) is the standard basis.

For \( t = 1, 2, \ldots: \)

With probability \( \gamma \), \( \triangleright \) exploration step

Choose \( i \in \{1, \ldots, n\} \) uniformly at random.

\( s_t := e_i \).

Play \( s_t \).

Observe \( \ell_t = c(s_t, w_t) \).

\( \tilde{w}_t := (\ell_t/\gamma) \Phi(e_i) \).

(\( \tilde{x}_{t+1}, \tilde{s}_{t+1} \) := APPROX-PROJ(\( \tilde{x}_t - \gamma \tilde{w}_t, \tilde{s}_t \),\( \tilde{x}_t \)) otherwise, with probability \( 1 - \gamma \), \( \triangleright \) exploitation step

\( s_t := s_t \).

Play \( s_t \).

Observe \( \ell_t = c(s_t, w_t) \).

(\( \tilde{x}_{t+1}, \tilde{s}_{t+1} \) := \( (x_t, s_t) \).

Figure 4: Algorithm for the bandit setting.
Taking expectations of both sides gives
\[
\sum_{t=1}^{T} x_t \cdot w_t - \alpha E \left[ \min_{x \in J} \sum_{t=1}^{T} x \cdot w_t \right] \leq \frac{(\alpha + 1)R^2}{2\eta} + T \frac{\delta}{\eta} + \eta T D^2 T.
\]
It thus suffices to show that
\[
E \left[ \min_{x \in J} \sum_{t=1}^{T} x \cdot \hat{w}_t \right] \geq \min_{x \in J} \sum_{t=1}^{T} x \cdot w_t - 2RD\sqrt{T}.
\]
(1)

Now, for any \( x \in J \),
\[
\left| \sum_{t=1}^{T} x \cdot (\hat{w}_t - w_t) \right| \leq |x| \sum_{t=1}^{T} |\hat{w}_t - w_t| \leq R \sum_{t=1}^{T} |\hat{w}_t - w_t|.
\]
This gives us a means of upper-bounding the difference between the minima. Namely,
\[
E \left[ \sum_{t=1}^{T} \hat{w}_t - w_t \right]^2 \leq E \left[ \left( \sum_{t=1}^{T} \hat{w}_t - w_t \right)^2 \right] = \sum_{t=1}^{T} E [ (\hat{w}_t - w_t)^2 ].
\]
(3)
The last equality follows from the fact that
\[
E[ (\hat{w}_t - w_t)^2 ] = 0
\]
for \( t_1 < t_2 \), which follows from the martingale-like assumption that \( E[ \hat{w}_{t_2} - w_{t_2} | \hat{w}_{t_1}, \hat{w}_{t_1} ] = 0 \). Finally,
\[
E[ (\hat{w}_t - w_t)^2 ] \leq E[ \hat{w}_t^2 + 2 \| \hat{w}_t \| \| w_t \| + w_t^2 ] \\
\leq D^2 + 2D^2 + 2D^2 = 4D^2.
\]
In the above we have used the facts that \( E[ \hat{w}_t^2 ] \leq E[ \hat{w}_t^2 ] \leq D^2 \) and \( \| w_t \|^2 \leq \| w_t \|^2 \\leq 2D^2 \). Hence, we have that the quantity in (3) is upper bounded by \( 4TD^2 \), which, together with (2), establishes (1).

4.1 Difficulty of the black-box reduction

We now point out that it is impossible to solve the bandit problem with general algorithms (approximation or exact) without an exploration basis (that is, if our only access to \( S \) is through a black-box optimization oracle). The counterexample is randomized. We will take
\[
W = \{ w \in \mathbb{R}^n \mid |w[1]| \leq 0, 1 \text{ and } \|w\|^2 \leq 2(w[1])^2 \}.
\]
The set \( S \) will consist of two points: \( s = (1/2, 0, \ldots, 0) \) as well as a second point \( s' = (0, \ldots, 0) - u \) where \( |u| = 1 \) and \( u[1] = 0 \). The mapping \( \Phi \) is the identity mapping.

The cost sequence for \( u_t = (1, 0, \ldots, 0) + v \) will be dependent on \( u \). Hence \( c(s, w_t) = 0 \) while \( c(s', w_t) = 0 \). Now, suppose we as algorithm designers know that this is the setup but \( u \) is chosen uniformly at random from the set of unit vectors with \( u[1] = 0 \).

\section*{Observation 4.3.}
For any bandit algorithm that makes \( k \) calls to black-box optimization oracle \( A \), any \( \alpha \geq 0 \), with probability \( 1 - ke^{-0.1\alpha} \) over \( u \), the algorithm has \( \alpha \)-regret 1/2 on a sequence of arbitrary length.

Proof. No information is conveyed by the costs returned in the bandit setup of our example—they are always 1/2 if \( s' \) has not been discovered, while the minimal cost is 0. Thus the algorithm must find some \( w \in W \) such that \( c(s, w) > c(s', w) \) (whence an exact optimization algorithm must return \( s' \)). Without loss of generality, we can scale \( w \) so that \( w[1] = 1 \) and \( \|w\| \leq 2 \). Hence, we can write \( w = (1, 0, \ldots, 0) + v \) where \( u[1] = 0 \) and \( |v| \leq 1 \). In this case, \( w \cdot s = 1/2 \), while \( w \cdot s' = 1 - u \cdot v \). For \( u \) a random unit vector and any fixed \( \|v\| \leq 1 \), it is known that \( \Pr[ u \cdot v \geq 1/2 ] \)
is exponentially small in \( n \). A very loose bound can be seen directly, since for a ball of dimension \( n \), this probability is

\[
\frac{\int_{0}^{1/2} (1-x^2)^{n-2}dx}{\int_{-1}^{1} (1-x^2)^{n-2}dx} \leq \frac{\int_{0}^{1/2} (3/4)^{-2}dx}{\int_{-1}^{1/2} (1-n^{-1})^{-2}dx} \leq \frac{\sqrt{\pi}}{\sqrt{2}} \left( \frac{3}{4} \right)^{n-1},
\]

which is \( O(e^{-0.1n}) \). \( \square \)

5. CONCLUSIONS AND OPEN PROBLEMS

We present a reduction converting approximate offline linear optimization problems into approximate online sequential linear optimization problems that holds for any approximation algorithm, in both the full-information setting and the bandit setting.

Our algorithm can be viewed as an analog to Hannan’s algorithm for playing repeated games against an unknown adversary. In our case, however, we cannot compute best responses but only approximately best responses.

The problem of obtaining similar results for interesting classes of non-linear optimization problems remains open.

6. REFERENCES


APPENDIX

A. EXAMPLE WHERE “FOLLOW-THE-LEADER” FAILS

First consider the set \( S = \{1, 2, \ldots, n\} \) and the cost sequence \((1, 1, \ldots, 1)\) (repeated \( T/n \) times), \((1, 0, 0, \ldots, 0)\) (repeated \( T/n \) times), \((0, 1, 0, \ldots, 0)\) (repeated \( T/n \) times), \((0, 0, 0, \ldots, 0)\) (repeated \( T/n \) times). Notice that a selection of decision, each period, which costs 1 is always a valid \( (\alpha = 2)\)-approximation to the leader on the previous examples. Moreover, its cost is \( T \) while the cost of the best (in fact every) \( s \in S \) is \( 2T/n \), hence giving large \( \alpha\)-regret. Unfortunately, adding perturbations of \( O(\sqrt{T}) \) as in follow-the-leader will not significantly improve matters. When \( T/n \gg \sqrt{T} \), a choice of decision which costs 1 each period is still an \( \alpha\)-approximation for, say, \( \alpha = 3 \).

Of course, one may be suspicious that no common approximation algorithms would have such peculiar behavior. We now give a similar example based on the standard greedy set cover approximation algorithm \( A (\alpha = \log m) \) applied to the online set cover problem described earlier. The example has \( n/2 \) covers of size 2: \( S_i = S \setminus S_{i+1} \), for \( i = 1, 2, \ldots, n \).

Furthermore, suppose the sets are of increasing size \( |S_i| = (0.4 + 0.2\frac{1}{n} - 1)m \) and \( |S_i \cup S_j| \leq 0.9m \) for all \( 1 \leq i, j \leq n \) where \( i \neq n+1-j \).\(^7\)

The sequence of costs (weight) vectors is divided into \( n/2 \) phases \( j = 0, 1, \ldots, n/2 - 1 \), each consisting of \( 2T/n \) identical cost vectors. In phase \( j = 0 \), all sets have cost 1. For phase \( j = 1, \ldots, n/2 - 1 \): the cost of the \( 2j-1 \) sets \( S_1, \ldots, S_{2j-1} \) and \( S_n, \ldots, S_{n-j+1} \) are all 1, while the costs of the remaining sets are all 0.

In this example, following the leader with greedy set cover will have an average per-period cost of at least 0.1. In particular, during the first 10% of any phase \( j \geq 1 \), either greedy’s first choice will be \( S_{n-j} \), in which case it’s second choice will be \( S_j \) (because any other set covers at most 90% of the remaining items, and \( S_j \)’s cost so far is at most 10% more than that of any other set), or greedy's first choice will be one of \( S_{n-j+1}, \ldots, S_n \); in either case it pays at least 1 during that period. Hence, following the leader pays at least \( 0.1 + \frac{0.1}{n} \) in expectation on average, while the cover \( S_{n/2} \cup S_{n/2+1} \) has an average cost of only \( 4/n \), which is far from matching greedy’s \( \alpha = \log m \) approximation ratio (for \( n = \theta(m) \)).

Also note that perturbations on the order of \( O(\sqrt{T}) \) will not solve this problem. It would be very interesting to adapt Hannan’s approach to work for approximation algorithms, especially because it is more efficient than our approach. However, we have not found a solution that works across problems.

\(^7\)To design such a collection of sets (for even \( n \) and \( m = 5(n - 1) \)), take \( S_i \) to be a uniformly random set of the desired size \( m \) for \( i = 1, \ldots, n/2 \), and \( S_{n-i+1} \) to be its complement. It is not hard to argue that, with high probability, the randomized construction obeys the stated properties.