

# Online Linear Optimization and Adaptive Routing

Baruch Awerbuch \*

Robert D. Kleinberg †

## Abstract

This paper studies an online linear optimization problem generalizing the *multi-armed bandit problem*. Motivated primarily by the task of designing adaptive routing algorithms for overlay networks, we present two randomized online algorithms for selecting a sequence of routing paths in a network with unknown edge delays varying adversarially over time. In contrast with earlier work on this problem, we assume that the only feedback after choosing such a path is the total end-to-end delay of the selected path. We present two algorithms whose regret is sublinear in the number of trials and polynomial in the size of the network. The first of these algorithms generalizes to solve any online linear optimization problem, given an oracle for optimizing linear functions over the set of strategies; our work may thus be interpreted as a general-purpose reduction from offline to online linear optimization. A key element of this algorithm is the notion of a *barycentric spanner*, a special type of basis for the vector space of strategies which allows any feasible strategy to be expressed as a linear combination of basis vectors using bounded coefficients.

We also present a second algorithm for the online shortest path problem, which solves the problem using a chain of online decision oracles, one at each node of the graph. This has several advantages over the online linear optimization approach. First, it is effective against an adaptive adversary, whereas our linear optimization algorithm assumes an oblivious adversary. Second, even in the case of an oblivious adversary, the second algorithm performs slightly better than the first, as measured by their additive regret.

## 1 Introduction

In a *multi-armed bandit problem*, the task of sequential decision-making under partial information is modeled as a repeated game between two parties (*algorithm* and *adversary*) interacting in a series of *trials*. The name derives from the metaphor of a gambler selecting slot machines in a casino, though overlay network routing is the most compelling motivation in the present context. Indeed, our primary motivation is the task of designing online algorithms for selecting a sequence of routing paths in a network with unknown link delays varying unpredictably over time. In contrast with earlier work on this problem (e.g. [10, 15]), we assume that the only feedback after choosing such a path is the total end-to-end delay of the selected path. Our algorithms thus face at least two distinct challenges:

---

\*Department of Computer Science, Johns Hopkins University, 3400 N. Charles Street, Baltimore MD 21218, USA. Email: [baruch@cs.jhu.edu](mailto:baruch@cs.jhu.edu). Supported by NSF grants ANIR-0240551 and CCR-0311795.

†Department of Mathematics, MIT, 77 Massachusetts Ave., Cambridge, MA 02139, USA. Email: [rdk@math.mit.edu](mailto:rdk@math.mit.edu). Supported by a Fannie and John Hertz Foundation Fellowship.

**Uncertainty about the past:** Each past trial reveals only the total end-to-end delay on one routing path in the network at one point in time. This provides only partial information about the link delays on the chosen path, and no information at all about other links in the network.

**Uncertainty about the future:** The past performance of links in the network is not assumed to be predictive of their future performance.

These two challenges are typical of multi-armed bandit problems. A key feature of algorithms for solving such problems is that they must balance *exploration* (experimenting with different alternatives to reveal information about their performance) versus *exploitation* (selecting the alternative which is currently believed to be best, with the objective of optimizing performance).

Existing work on multi-armed bandit problems (e.g. [2]) assumes the algorithm chooses from a fixed set of  $K$  alternatives, or “strategies,” in each trial. The online learning algorithms studied in this prior work outperform random guessing only when the number of trials is at least  $\Omega(K \log(K))$ . Such algorithms are impractical when the number of strategies,  $K$ , is exponential in the problem size, for instance when  $K$  is the number of routing paths in a designated network. Here, we design algorithms for generalized multi-armed bandit problems in which the set of strategies may be exponentially or even infinitely large, but may be embedded in a low-dimensional vector space such that the cost functions are represented by linear functions. Our algorithms are efficient in the sense that the average cost of the strategies chosen by the algorithm approximates the average cost of the best single strategy — up to an arbitrarily small additive constant — even when the number of trials is only polynomial in the problem size. To obtain such online algorithms, we need only assume access to an oracle for minimizing a linear function on the set of alternatives. Our work may thus be interpreted as a general-purpose reduction from offline to online linear optimization.

## 1.1 Problem formulation

We adopt the following framework for generalized multi-armed bandit problems. Special cases have been studied in earlier works, e.g. [1, 2].

- One is given a set  $\mathcal{S}$  and a set  $\Gamma$  of cost functions mapping  $\mathcal{S}$  to  $\mathbb{R}$ . We refer to elements of  $\mathcal{S}$  as “strategies” and elements of  $\Gamma$  as “cost functions.” One is also given an upper bound  $T$  on the number of trials.<sup>1</sup>
- An algorithm is a (possibly randomized) rule for choosing an element  $x_j \in \mathcal{S}$ , for each trial  $j \in \{1, 2, \dots, T\}$ , given a sequence of ordered pairs  $(x_i, y_i)$ ,  $1 \leq i \leq j - 1$ , representing the strategies and costs observed in previous trials.
- An adversary is a (possibly randomized) rule for choosing an element  $c_j \in \Gamma$ , for each trial  $j \in \{1, 2, \dots, T\}$ , given a sequence of strategies  $x_i$ ,  $1 \leq i \leq j - 1$ , representing the strategies chosen by the algorithm in previous trials.

---

<sup>1</sup>All of the algorithms considered in this paper can also be made to work without foreknowledge of  $T$ , by using a standard doubling technique in which the algorithm initially guesses that  $T = 1$ , then doubles its guess and re-initializes its state every time the number of trials exceeds the value of the current guess. This worsens the bounds derived in the paper by only a constant factor.

- An algorithm’s *regret* against a specified adversary is the difference in expected cost between the algorithm’s sequence of choices and that of the best fixed strategy in  $\mathcal{S}$ , i.e.,

$$\text{Regret} = \mathbf{E} \left[ \sum_{j=1}^T c_j(x_j) \right] - \min_{x \in \mathcal{S}} \sum_{j=1}^T \mathbf{E} [c_j(x)].$$

The algorithm’s overall regret is defined as the maximum of the regret against all adversaries.

The following two variations of the model are of interest.

**Transparent feedback** means that the entire cost function  $c_i$  is revealed, in contrast to the *opaque* feedback model specified above, in which only the cost of the chosen strategy is revealed. In other words, the transparent feedback model assumes that the algorithm’s choice of  $x_j$  is allowed to depend on the pairs  $(x_i, c_i)$  where  $c_i$  represents the cost function chosen by the adversary at time  $i$ , rather than the opaque feedback model in which  $x_j$  depends only on the pairs  $(x_i, y_i)$  where  $y_i$  represents the value  $c_i(x_i)$ . In the online learning theory literature, the transparent and opaque models are referred to as the “best expert” [13] and “multi-armed bandit” problems [2], respectively.

An **oblivious adversary** (in contrast to an adaptive adversary) does not see the algorithm’s past decisions. In other words, an oblivious adversary is simply a sequence of cost functions  $c_1, \dots, c_T \in \Gamma$ , chosen “up front” without any dependence on the algorithm’s random choices  $x_1, x_2, \dots, x_T$ .

The traditional multi-armed bandit problem (e.g. [2]) corresponds to the opaque feedback model with strategy set  $\mathcal{S} = \{1, 2, \dots, K\}$  and cost function set  $\Gamma = [0, 1]^{\mathcal{S}}$  consisting of all  $[0, 1]$ -valued functions on  $\mathcal{S}$ .<sup>2</sup> We focus here on two generalizations of this problem.

- *Dynamic shortest path*: The strategy set  $\mathcal{S}$  consists of paths from a sender  $s$  to a receiver  $r$  in a directed graph  $G = (V, E)$ . A cost function  $c \in \Gamma$  is specified by assigning lengths in  $[0, 1]$  to the edges of  $G$ . The cost assigned to a path  $\pi$  by such a function is the sum of its edge lengths. The algorithm’s goal is to nearly match the cost of a shortest path of at most  $H$  edges from  $s$  to  $r$ .
- *Online linear optimization*: In this more general problem, the strategy set  $\mathcal{S}$  is a compact subset of  $\mathbb{R}^d$ , and the set  $\Gamma$  of cost functions consists of all linear functions mapping  $\mathcal{S}$  to  $[-M, M]$ , for some constant  $M$ .

The dynamic shortest path problem may be applied to overlay network routing, by interpreting edge costs as link delays. In our formulation, the feedback from each trial is limited to exposing the end-to-end delay from sender to receiver; this models the notion that no feedback is obtained from intermediate routers in the network.

## 1.2 Our contributions

We present two algorithms, both in the opaque feedback model:

---

<sup>2</sup>The problem is typically formulated as a maximization problem, rather than a minimization problem as here. The difference is immaterial from our standpoint, since we may transform a maximization problem into a minimization problem by replacing each function  $c$  with the function  $1 - c$ .

1. An online linear optimization algorithm, achieving regret  $O(T^{2/3}Md^{5/3})$  against an oblivious adversary, where  $d$  is the dimension of the problem space.
2. A dynamic shortest path algorithm, achieving regret  $O(T^{2/3}H^{7/3}(m \log \Delta \log(m \cdot T))^{1/3})$  against an adaptive adversary, in a graph with  $m$  edges and maximum degree  $\Delta$ , when the path length is constrained to be at most  $H$  hops.

Since the dynamic shortest path problem is a special case of online linear optimization, it is also possible to use the first algorithm as a dynamic shortest path algorithm. Here we have  $d = m - n + 2$  (the dimension of the vector space of  $s - r$  flows) and  $M = H$ , leading to a weaker regret bound than that achieved by the second algorithm (except in cases when  $(\frac{m}{H})^4 < \log \Delta \log(m \cdot T)$ ), and in a weaker adversarial model. As in [2], the challenge in both algorithms is to deal with the incomplete (i.e. opaque) feedback. However, even the full-feedback versions of these problems do not allow greedy solutions and require the framework of randomized learning algorithms [10, 13, 15].

For the online linear optimization problem, a novel idea in our work is to compute a special basis for the vector space spanned by the strategy set. This basis, which is called a *barycentric spanner*, has the property that all other strategies can be expressed as linear combinations *with bounded coefficients* of the basis elements. Against an oblivious adversary, we can sample basis elements occasionally, use linear interpolation to estimate the costs of all other strategies, and feed these estimates into the transparent-feedback online linear optimization algorithm of Kalai and Vempala [10], which achieves small regret and polynomial computation time with the help of an offline linear optimization oracle. We also provide a polynomial-time construction of the barycentric spanner, using the same optimization oracle as a subroutine.

Our dynamic shortest path algorithm, which works against an adaptive adversary, can be viewed as a distributed learning algorithm with a “local learner” at each node in the graph, making decisions about which outgoing edge lies on the shortest path towards the receiver  $r$ . Ideally, each node would know the past costs of each outgoing edge. However, this information is not available; instead the local learner is only given the total end-to-end path cost, for all prior sample paths passing through that node. The local learner can now try to correlate past decisions with the overall quality of the paths obtained, and can notice that some outgoing edges appear to lead to better global decisions than other outgoing edges. What makes this judgement quite confusing is that the quality of such decisions depends on the decisions of downstream and upstream nodes, as well as on the decisions of the adaptive adversary, who has knowledge of the algorithm and of its past random choices. Thus, the adversary has significant power to confuse the measurements of the local learner, and it is somewhat surprising that there exists an online algorithm which successfully defeats such an adversary.

### 1.3 Comparison with existing work

Online learning problems of the sort formulated in Section 1.1 have been studied for many years. The special case of a small finite strategy set  $S = \{1, 2, \dots, K\}$  and cost function set  $\Gamma = [0, 1]^S$  has received the most attention in prior work. The transparent-feedback case of this problem is commonly known as the *best-expert* problem because of the metaphor of learning from the advice of  $K$  experts. Littlestone and Warmuth’s seminal weighted-majority algorithm [13] achieves  $O(\sqrt{T \log K})$  regret in

the best-expert problem. The opaque-feedback case of the same problem is commonly known as the *multi-armed bandit* problem because of the metaphor of a gambler selecting slot machines in a casino. The majority of work on this problem (e.g. [7, 9, 12]) deals with the case when cost functions are independent, identically distributed samples from a distribution on  $\Gamma$ , but Auer et al [2] have studied multi-armed bandit problems in the same adversarial model considered here, obtaining an algorithm with regret  $O(\sqrt{TK \log K})$ . Note that for the shortest paths problem, the cardinality of the strategy set may be exponential in the size of the graph  $G$  (i.e.  $K = 2^{\Omega(n)}$ ). On some simple instances, e.g. a chain of  $n$  vertices with two parallel edges between each pair of vertices in the chain, the algorithms of [2] do in fact exhibit exponential regret.

Online learning problems with larger strategy sets — particularly the dynamic shortest path problem — have received attention in recent years, but algorithms prior to ours either assumed transparent feedback or they assumed a feedback model which is intermediate between transparent and opaque feedback. Takimoto and Warmuth [15] studied the dynamic shortest path problem with transparent feedback, demonstrating that the weighted-majority algorithm in this case can be simulated by an efficient (polynomial-time) algorithm in spite of the fact that there are exponentially many “experts”, corresponding to all the  $s - r$  paths in  $G$ . Kalai and Vempala [10] considered online linear optimization for a strategy set  $\mathcal{S} \subseteq \mathbb{R}^d$ , presenting an algorithm which achieves  $O(\sqrt{T \log d})$  regret. (Note that this bound depends only on the dimension  $d$ , hence it is applicable even if the cardinality of  $\mathcal{S}$  is exponential or infinite.) One may interpret this as an algorithm for the dynamic shortest path problem, by considering the set of  $s - r$  paths in  $G$  as the vertices of the polytope of unit flows from  $s$  to  $r$ ; thus Kalai and Vempala’s algorithm also constitutes a dynamic shortest path algorithm with regret  $O(\sqrt{T \log m})$  in the transparent feedback model.

Awerbuch and Mansour [5] considered an online path-selection problem in a model where the edges have binary costs (they either fail or they do not fail) and the cost of a path is 1 if any edge fails, 0 otherwise. They consider a “prefix” feedback model, where the feedback in each trial identifies the location of the *first* edge failure, if any edge failed. Using the best-expert algorithm [13] as a black box, their algorithm obtains regret  $O(H(n \log(nT))^{1/2} T^{5/6})$  assuming an oblivious adversary. The algorithm was strengthened to work against an adaptive adversary in subsequent work by Awerbuch, Holmer, Rubens, and Kleinberg [3].<sup>3</sup> Our Algorithm 2 is structurally similar to these two algorithms, in that it proceeds in phases and involves local decision-making at the nodes using the best-expert algorithm. However, [3, 5] both require richer feedback, as described earlier. The novelty of our solution is the introduction of a somewhat non-intuitive “prefix” probability distribution, schematically illustrated in Figure 4. Sampling from this distribution enables us to prove that the algorithm is robust against an adaptive adversary, achieving regret which is sublinear in  $T$  and polynomial in the size of the graph.

Subsequent to the publication of an extended abstract of our work in [4], several papers have introduced enhancements or extensions of the results presented here. McMahan and Blum [14] modified Algorithm 1 to make it robust against an adaptive adversary, achieving regret  $O(T^{3/4} \sqrt{\log T})$  for online linear optimization in the opaque feedback model.<sup>4</sup> The online linear optimization problem with

<sup>3</sup>These two results are based on a somewhat different notion of “regret” from that considered here; see [3, 5] for details.

<sup>4</sup>The regret of the algorithm also depends polynomially on  $d$ , the dimension of the problem space; see [14] for details.

a convex strategy set  $\mathcal{S}$  is a special case of *online convex optimization*, in which  $\mathcal{S}$  is a convex subset of  $\mathbb{R}^d$  and  $\Gamma$  is the set of convex functions mapping  $\mathcal{S}$  to a bounded interval. This problem was solved in the transparent-feedback model by Zinkevich [16], and the opaque-feedback version was considered independently by Kleinberg [11] and by Flaxman, Kalai, and McMahan [8]. The former paper presents an algorithm achieving regret  $O(d^{13/4}n^{3/4})$  against an oblivious adversary, assuming the cost functions are twice-differentiable, with bounded first and second derivatives. As in our Algorithm 1, the key idea in [11] is to use barycentric spanners to supply an efficient reduction from the opaque-feedback problem to the corresponding transparent-feedback problem. The algorithm of [8] is also based on reducing from the opaque-feedback problem to the transparent-feedback problem, but their reduction uses a clever one-point estimate of the gradient instead of using a barycentric spanner. Their analysis also allows an adaptive adversary, requires no smoothness hypothesis on the cost functions, and achieves a stronger regret bound of  $O(dn^{3/4})$ .

## 2 Online linear optimization

### 2.1 Overview of algorithm

This section presents a randomized algorithm for online linear optimization, in which the strategy set  $S$  is a compact subset of  $\mathbb{R}^d$  and the cost functions are linear functions mapping  $S$  to  $[-M, M]$  for some predefined constant  $M$ . As stated in the introduction, the transparent-feedback version of this problem has been solved by Kalai and Vempala in [10]. We will use their algorithm as a black box (the *K-V black box*), reducing from the opaque-feedback case to the transparent-feedback case by dividing the timeline into phases and using each phase to simulate one round of the transparent-feedback problem. We randomly subdivide the time steps in a phase into a small number of “exploration” steps which are used for explicitly sampling the costs of certain strategies, and a much larger number of “exploitation” steps in which we choose our strategy according to the output of the black box, with the aim of minimizing cost. The feedback to the black box at the end of a phase is an unbiased estimate of the average of the cost vectors in that phase, generated by averaging the data from the sampling steps. (Ideally, we would also use the data from the exploitation steps, since it is wasteful to throw this data away. However, we do not know how to incorporate this data without biasing our estimate of the average cost function. This shortcoming of the analysis partly explains why we are limited, in this section, to considering the oblivious adversary model.)

We now address the question of how to plan the sampling steps so as to generate a reasonably accurate and unbiased estimate of the average cost vector in a phase. One’s instinct, based on the multi-armed bandit algorithm of [2], might be to try sampling each strategy a small percentage of the time, and to ascribe to each strategy a simulated cost which is the average of the samples. The problem with this approach in our context is that there may be exponentially many, or even infinitely many, strategies to sample. So instead we take advantage of the fact that the cost functions are linear, to sample a small subset  $X \subseteq S$  of the strategies — a basis for the vector space spanned by  $S$  — and extend the simulated cost function from  $X$  to  $S$  by linear interpolation. In taking this approach, a subtlety arises which accounts for the main technical contribution of this section. The problem is that the average of the sampled costs at a point of  $X$  will generally differ from the true average cost by a

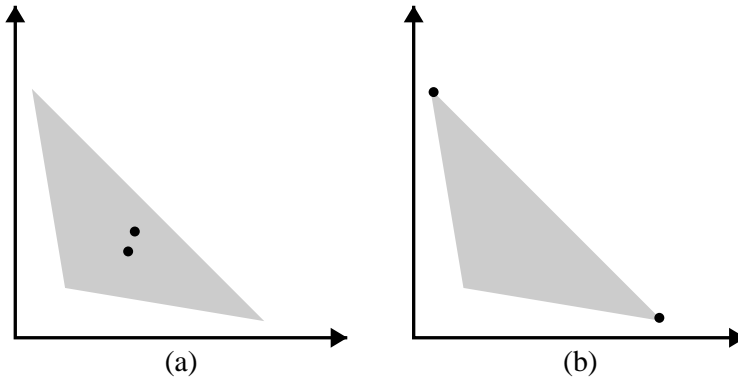


Figure 1: (a) A bad sampling set (b) A barycentric spanner.

small sampling error; if the point set  $X$  is badly chosen, these sampling errors will be amplified by an arbitrarily large factor when we extend the simulated cost function to all of  $S$ . (See Figure 2.1. In that example,  $S$  is a triangle in  $\mathbb{R}^2$ . The point set on the left is bad choice for  $X$ , since small sampling errors can lead to large errors at the upper left and lower right corners. The point set on the right does not suffer from this problem.)

To avoid this pitfall, we must choose  $X$  to be as “well-spaced” inside  $S$  as possible. We formulate this notion of “well-spaced subsets” precisely in Section 2.2; such a subset will be called a *barycentric spanner*. Using barycentric spanners, we give a precise description and analysis of the online linear optimization algorithm sketched above. We then illustrate how these techniques may be applied to the dynamic shortest path problem.

## 2.2 Barycentric spanners

**Definition 2.1.** Let  $V$  be a vector space over the real numbers, and  $S \subseteq V$  a subset whose linear span is a  $d$ -dimensional subspace of  $V$ . A set  $X = \{x_1, \dots, x_d\} \subseteq S$  is a *barycentric spanner* for  $S$  if every  $x \in S$  may be expressed as a linear combination of elements of  $X$  using coefficients in  $[-1, 1]$ .  $X$  is a  $C$ -approximate barycentric spanner if every  $x \in S$  may be expressed as a linear combination of elements of  $X$  using coefficients in  $[-C, C]$ .

**Proposition 2.2.** *If  $S$  is a compact subset of  $V$ , then  $S$  has a barycentric spanner.*

*Proof.* Assume without loss of generality that  $\text{span}(S) = V = \mathbb{R}^d$ . Choose a subset  $X = \{x_1, \dots, x_d\} \subseteq S$  maximizing  $|\det(x_1, \dots, x_d)|$ . (The maximum is attained by at least one subset of  $S$ , by compactness.) We claim  $X$  is a barycentric spanner of  $S$ . For any  $x \in S$ , write  $x = \sum_{i=1}^d a_i x_i$ . Then

$$|\det(x, x_2, x_3, \dots, x_d)| = \left| \det \left( \sum_{i=1}^d a_i x_i, x_2, x_3, \dots, x_d \right) \right|$$

$$\begin{aligned}
&= \left| \sum_{i=1}^d a_i \det(x_i, x_2, x_3, \dots, x_d) \right| \\
&= |a_1| |\det(x_1, \dots, x_d)|
\end{aligned}$$

from which it follows that  $|a_1| \leq 1$ , by the maximality of  $|\det(x_1, \dots, x_d)|$ . By symmetry, we see that  $|a_i| \leq 1$  for all  $i$ , and we conclude (since  $x$  was arbitrary) that  $X$  is a barycentric spanner as claimed.  $\square$

**Observation 2.3.** Given a subset  $X = \{x_1, \dots, x_d\} \subseteq S$  and an index  $i \in \{1, \dots, d\}$ , let  $X_{-i}$  denote the  $(d-1)$ -tuple of vectors  $(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_d)$ . The proof of Proposition 2.2 actually establishes the following stronger fact. If  $X = \{x_1, \dots, x_d\}$  is a subset of  $S$  with the property that for any  $x$  in  $S$  and any  $i \in \{1, \dots, d\}$ ,

$$|\det(x, X_{-i})| \leq C |\det(x_1, x_2, \dots, x_d)|,$$

then  $X$  is a  $C$ -approximate barycentric spanner for  $S$ .

A consequence of Proposition 2.2 is the following matrix factorization theorem, which was independently proven by Barnett and Srebro [6]. For a matrix  $M = (m_{ij})$ , let  $\|M\|_\infty = \max_{i,j} |m_{ij}|$ .

**Proposition 2.4.** *If  $M$  is an  $m$ -by- $n$  matrix satisfying  $\|M\|_\infty \leq 1$  and  $\text{rank}(M) = k$ , then we may write  $M$  as a product  $M = AB$  where  $A, B$  are  $m$ -by- $k$  and  $k$ -by- $n$  matrices, respectively, satisfying  $\|A\|_\infty \leq 1$  and  $\|B\|_\infty \leq 1$ .*

Interestingly, Barnett and Srebro’s proof of Proposition 2.4 is non-constructive, making use of Kakutani’s fixed point theorem, just as our proof of Proposition 2.2 is non-constructive, relying on minimizing the function  $|\det(x_1, \dots, x_n)|$  on the compact set  $S^d$ . In fact, it is an open question whether barycentric spanners can be computed in polynomial time, given an oracle for optimizing linear functions over  $S$ . However, the following proposition shows that  $C$ -approximate barycentric spanners (for any  $C > 1$ ) may be computed in polynomial time given access to such an oracle.

**Proposition 2.5.** *Suppose  $S \subseteq \mathbb{R}^d$  is a compact set not contained in any proper linear subspace. Given an oracle for optimizing linear functions over  $S$ , for any  $C > 1$  we may compute a  $C$ -approximate barycentric spanner for  $S$  in polynomial time, using  $O(d^2 \log_C(d))$  calls to the optimization oracle.*

*Proof.* The algorithm is shown in Figure 2. Here, as elsewhere in this paper, we sometimes follow the convention of writing a matrix as a  $d$ -tuple of column vectors. The matrix  $(\mathbf{e}_1, \dots, \mathbf{e}_d)$  appearing in the first step of the algorithm is the identity matrix. The “for” loop in the first half of the algorithm transforms this into a basis  $(x_1, x_2, \dots, x_d)$  contained in  $S$ , by replacing the original basis vectors  $(\mathbf{e}_1, \dots, \mathbf{e}_d)$  one-by-one with elements of  $S$ . Each iteration of the loop requires two calls to the optimization oracle, to compute  $x_0 := \arg \max_{x \in S} |\det(x, X_{-i})|$  by comparing the maxima of the linear functions

$$\ell_i(x) = \det(x, X_{-i}), \quad -\ell_i(x) = -\det(x, X_{-i}).$$

This  $x_0$  is guaranteed to be linearly independent of the vectors in  $X_{-i}$  because  $\ell_i$  evaluates to zero on  $X_{-i}$ , and is nonzero on  $x_0$ . ( $\ell_i$  is non-zero on at least one point  $x \in S$  because  $S$  is not contained in a proper subspace of  $\mathbb{R}^d$ .)

```

/* First, compute a basis of  $\mathbb{R}^d$  contained in  $S$ . */
 $(x_1, \dots, x_d) \leftarrow (\mathbf{e}_1, \dots, \mathbf{e}_d)$ ;
for  $i = 1, 2, \dots, d$  do
    /* Replace  $x_i$  with an element of  $S$ ,
       while keeping it linearly independent from  $X_{-i}$ . */
     $x_i \leftarrow \arg \max_{x \in S} |\det(x, X_{-i})|$ ;
end

/* Transform basis into approximate barycentric spanner. */
while  $\exists x \in S, i \in \{1, \dots, d\}$  satisfying
     $|\det(x, X_{-i})| > C |\det(x_i, X_{-i})|$ 
     $x_i \leftarrow x$ ;
end
return  $(x_1, x_2, \dots, x_d)$ 

```

Figure 2: Algorithm for computing a  $C$ -approximate barycentric spanner.

Lemma 2.6 below proves that the number of iterations of the “while” loop in the second half of the algorithm is  $O(d \log_C(d))$ . Each such iteration requires at most  $2d$  calls to the optimization oracle, i.e. two to test the conditional for each index  $i \in \{1, \dots, d\}$ . At termination,  $(x_1, \dots, x_d)$  is a  $C$ -approximate barycentric spanner, by Observation 2.3.  $\square$

**Lemma 2.6.** *The total number of iterations of the “while” loop is  $O(d \log_C(d))$ .*

*Proof.* Let  $M_i = (x_1, x_2, \dots, x_i, \mathbf{e}_{i+1}, \dots, \mathbf{e}_d)$  be the matrix whose columns are the basis vectors at the end of the  $i$ -th iteration of the “for” loop. (Columns  $i + 1$  through  $d$  are unchanged at this point in the algorithm.) Let  $M = M_d$  be the matrix at the end of the “for” loop, and let  $M'$  be the matrix at the end of the algorithm. Henceforth in this proof,  $(x_1, \dots, x_d)$  will refer to the columns of  $M$ , not  $M'$ .

It suffices to prove that  $|\det(M')/\det(M)| \leq d^{d/2}$ , because the determinant of the matrix increases by a factor of at least  $C$  on each iteration of the “while” loop. Let  $U$  be the matrix whose  $i$ -th row is  $u_i := \mathbf{e}_i^\top M_i^{-1}$ , i.e. the  $i$ -th row of  $M_i^{-1}$ . Recalling the linear function  $\ell_i(x) = \det(x, X_{-i})$ , one may verify that

$$u_i x = \frac{\ell_i(x)}{\ell_i(x_i)} \quad \forall x \in \mathbb{R}^d, \quad (1)$$

by observing that both sides are linear functions of  $x$  and that the equation holds when  $x$  is any of the columns of  $M_i$ . It follows that  $|u_i x| \leq 1$  for all  $x \in S$ , since  $x_i = \arg \max_{x \in S} |\ell_i(x)|$ . Each entry of the matrix  $UM'$  is equal to  $u_i x$  for some  $i \in \{1, \dots, d\}$ ,  $x \in S$ , so the entries of  $UM'$  lie between  $-1$  and  $1$ . Hence  $|\det(UM')| \leq d^{d/2}$ . (The determinant of a matrix cannot exceed the product of the  $L^2$ -norms of its columns.) Again using equation (1), observe that  $u_i x_j$  is equal to 0 if  $j < i$ , and is equal to 1 if  $j = i$ . In other words  $UM$  is an upper triangular matrix with 1’s on the diagonal. Hence  $\det(UM) = 1$ . Now

$$\left| \frac{\det(M')}{\det(M)} \right| = \left| \frac{\det(UM')}{\det(UM)} \right| \leq d^{d/2},$$

as desired. □

### 2.3 The online linear optimization algorithm

Our algorithm will employ a subroutine known as the “Kalai-Vempala algorithm with parameter  $\varepsilon$ .” (Henceforth the “K-V black box.”) The K-V black box is initialized with a parameter  $\varepsilon > 0$  and a set  $S \subseteq \mathbb{R}^d$  of strategies. It receives as input a sequence of linear cost functions  $c_j : S \rightarrow \mathbb{R}$ , ( $1 \leq j \leq t$ ), taking values in  $[-M, M]$ . Given a linear optimization oracle for  $S$ , it computes a sequence of probability distributions  $p_j$  on  $S$ , such that  $p_j$  depends only on  $c_1, c_2, \dots, c_{j-1}$ . The K-V black box meets the following performance guarantee: if  $x^{(1)}, \dots, x^{(t)}$  are random samples from  $p_1, \dots, p_t$ , respectively, and  $x$  is any point in  $S$ , then

$$\mathbf{E} \left[ \frac{1}{t} \sum_{j=1}^t c_j(x^{(j)}) \right] \leq O(\varepsilon M d^2 + M d^2 / \varepsilon t) + \frac{1}{t} \sum_{j=1}^t c_j(x). \quad (2)$$

See [10] for a description and analysis of the Kalai-Vempala algorithm with parameter  $\varepsilon$ . Their paper differs from ours in that they assume the cost vectors satisfy  $\|c_j\|_1 \leq 1$ , and they express the regret bound as

$$\mathbf{E} \left[ \frac{1}{t} \sum_{j=1}^t c_j(x^{(j)}) \right] \leq D \left( \frac{\varepsilon}{2} + \frac{1}{\varepsilon t} \right) + \frac{1}{t} \sum_{j=1}^t c_j(x), \quad (3)$$

where  $D$  is the  $L^1$ -diameter of  $S$ . To derive (2) from this, let  $\{x_1, \dots, x_d\}$  be a 2-approximate barycentric spanner for  $S$ , and transform the coordinate system by mapping  $x_i$  to  $(Md)\mathbf{e}_i$ , for  $i = 1, \dots, d$ . This maps  $S$  to a set whose  $L^1$ -diameter satisfies  $D \leq 4Md^2$ , by the definition of a 2-approximate barycentric spanner. The cost vectors in the transformed coordinate system have no component whose absolute value is greater than  $1/d$ , hence they satisfy the required bound on their  $L^1$ -norms.

Our algorithm precomputes a 2-approximate barycentric spanner  $X \subseteq S$ , and initializes an instance of the K-V black box with parameter  $\varepsilon$ , where  $\varepsilon = (dT)^{-1/3}$ . Assume, for simplicity, that  $T$  is divisible by  $d^2$  and that  $T/d^2$  is a perfect cube.<sup>5</sup> Divide the timeline  $1, 2, \dots, T$  into phases of length  $\tau = d/\delta$ , where  $\delta = \varepsilon d^2$ ; note that  $\tau$  is an integer by our assumption on  $T$ . The time steps in phase  $\phi$  are numbered  $\tau(\phi - 1) + 1, \tau(\phi - 1) + 2, \dots, \tau\phi$ . Call this set of time steps  $\mathcal{T}_\phi$ . Within each phase, the algorithm selects a subset of  $d$  time steps uniformly at random, and chooses a random one-to-one correspondence between these time steps and the elements of  $X$ . The step in phase  $\phi$  corresponding to  $x_i \in X$  will be called the “sampling step for  $x_i$  in phase  $\phi$ ,” all other time steps will be called “exploitation steps.” In a sampling step for  $x_i$ , the algorithm chooses strategy  $x_i$ ; in an exploitation step it samples its strategy randomly using the probability distribution computed by the K-V black box.

At the end of each phase, the algorithm updates its K-V black box algorithm by feeding in the unique cost vector  $c_\phi$  such that, for all  $i \in \{1, \dots, d\}$ ,  $c_\phi \cdot x_i$  is equal to the cost observed in the sampling step for  $x_i$ .

---

<sup>5</sup>If  $T$  does not satisfy these properties, we may replace  $T$  with another integer  $T' = O(T + d^2)$  without affecting the stated bounds by more than a constant factor.

**Theorem 2.7.** *The algorithm achieves regret of  $O(Md^{5/3}T^{2/3})$  against an oblivious adversary, where  $d$  is the dimension of the problem space.*

*Proof.* Note that the cost vector  $c_\phi$  satisfies  $|c_\phi \cdot x_i| \leq M$  for all  $x_i \in X$ , and that its expectation is

$$\bar{c}_\phi = \mathbf{E}[c_\phi] = \frac{1}{\tau} \sum_{j \in \mathcal{T}_\phi} c_j.$$

Let  $t = T/\tau$ ; note that  $t$  is an integer by our assumption on  $T$ . The performance guarantee for the K-V algorithm ensures that for all  $x \in S$ ,

$$\mathbf{E} \left[ \frac{1}{t} \sum_{\phi=1}^t c_\phi \cdot x_\phi \right] \leq O \left( \varepsilon M d^2 + \frac{M d^2}{\varepsilon t} \right) + \frac{1}{t} \sum_{\phi=1}^t c_\phi \cdot x, \quad (4)$$

where  $x_\phi$  is a random sample from the probability distribution specified by the black box in phase  $\phi$ . Henceforth we will denote the term  $O(\varepsilon M d^2 + M d^2/\varepsilon t)$  on the right side by  $R$ . Now let's take the expectation of both sides with respect to the algorithm's random choices. The key observation is that  $\mathbf{E}[c_\phi \cdot x_j] = \mathbf{E}[\bar{c}_\phi \cdot x_j]$ . This is because  $c_\phi$  and  $x_j$  are independent random variables:  $c_\phi$  depends only on sampling decisions made by the algorithm in phase  $\phi$ , while  $x_j$  depends only on data fed to the K-V black box before phase  $\phi$ , and random choices made by the K-V box during phase  $\phi$ . Hence

$$\mathbf{E}[c_\phi \cdot x_j] = \mathbf{E}[c_\phi] \cdot \mathbf{E}[x_j] = \bar{c}_\phi \cdot \mathbf{E}[x_j] = \mathbf{E}[\bar{c}_\phi \cdot x_j].$$

Now taking the expectation of both sides of (4) with respect to the random choices of both the algorithm and the black box, we find that for all  $x \in S$ ,

$$\begin{aligned} \mathbf{E} \left[ \frac{1}{t} \sum_{\phi=1}^t \bar{c}_\phi \cdot x_\phi \right] &\leq R + \frac{1}{t} \sum_{\phi=1}^t \bar{c}_\phi \cdot x \\ \mathbf{E} \left[ \frac{1}{t\tau} \sum_{\phi=1}^t \sum_{j \in \mathcal{T}_\phi} c_j \cdot x_j \right] &\leq R + \frac{1}{t\tau} \sum_{\phi=1}^t \sum_{j \in \mathcal{T}_\phi} c_j \cdot x \\ \mathbf{E} \left[ \frac{1}{T} \sum_{j=1}^T c_j \cdot x_j \right] &\leq R + \frac{1}{T} \sum_{j=1}^T c_j \cdot x \\ \mathbf{E} \left[ \sum_{j=1}^T c_j \cdot x_j \right] &\leq RT + \sum_{j=1}^T c_j \cdot x. \end{aligned}$$

The left side is an upper bound on the total expected cost of all exploitation steps. The total cost of all sampling steps is at most  $Mdt = \delta MT$ . Thus the algorithm's expected regret satisfies

$$\begin{aligned} \text{Regret} &\leq RT + \delta MT \\ &= O \left( \varepsilon M d^2 T + \frac{M d^2 T}{\varepsilon t} + \delta MT \right) \\ &= O \left( (\delta + \varepsilon d^2) MT + \frac{M d^3}{\varepsilon \delta} \right). \end{aligned}$$

Recalling that  $\varepsilon = (dT)^{-1/3}$ ,  $\delta = d^{5/3}T^{-1/3}$ , we obtain

$$\text{Regret} = O(T^{2/3}Md^{5/3}).$$

□

## 2.4 Application to dynamic shortest path problem

To apply this algorithm to the dynamic shortest path problem, the vector space  $\mathbb{R}^d$  will be the space of all flows from  $s$  to  $r$  in  $G$ , i.e. the linear subspace of  $\mathbb{R}^m$  satisfying the flow conservation equations at every vertex except  $s, r$ . (Thus  $d = m - n + 2$ .) The set  $S$  of all paths of length at most  $H$  from  $s$  to  $r$  is embedded in  $\mathbb{R}^d$  by associating each path with the corresponding unit flow. Specifying a set of edge lengths defines a linear cost function on  $\mathbb{R}^d$ , namely the function which assigns to each flow the weighted sum of the lengths of all edges used by that flow, weighted by the amount of flow traversing the edge. The linear optimization oracle over  $S$  may be implemented using a suitable shortest-path algorithm, such as Bellman-Ford. The algorithm in Figure 2 describes how to compute a set of paths which form a 2-approximate barycentric spanner for  $S$ . Applying the bound on regret from section 2.3, we obtain

$$\text{Regret} = O(T^{2/3}Hm^{5/3}).$$

We should mention that maximal linearly-independent sets of paths are not always approximate barycentric spanners. In fact, there are examples of graphs of size  $n$  having a maximal linearly-independent set of paths which is not a  $C$ -approximate barycentric spanner for any  $C = 2^{o(n)}$ .

## 3 Dynamic shortest paths

### 3.1 The model

Recall that in the dynamic shortest path problem considered here, one is given a directed graph  $G = (V, E)$ . For  $j = 1, 2, \dots, T$ , an adversary selects a cost function  $c_j : E \rightarrow [0, 1]$ . The adversary is adaptive, in that  $c_j$  may depend on the algorithm's choices in previous time steps. The online algorithm must select a (not necessarily simple) path of length less than or equal to  $H$  from a fixed source  $s$  to a fixed receiver  $r$ , receiving as feedback the cost of this path, defined as the sum of  $c_j(e)$  for all edges  $e$  on the path. Our goal is to minimize the algorithm's regret, i.e. the difference between the algorithm's expected total cost and the total cost of the best single path from  $s$  to  $r$ . (The "best path" here refers to a path of minimum expected cost.)

### 3.2 Algorithm Structure

As in [5], we begin by transforming our arbitrary input graph  $G = (V, E)$  with  $n$  vertices and  $m$  edges into a levelled directed acyclic graph  $\tilde{G} = (\tilde{V}, \tilde{E})$ . This graph  $\tilde{G}$  is a subgraph of the graph  $G \times \{0, 1, \dots, H\}$  which has vertex set  $V \times \{0, 1, \dots, H\}$  and which has an edge  $e_i$  from  $(u, i - 1)$  to

$(v, i)$  for every edge  $e = (u, v) \in E$  and every  $i \in \{1, \dots, H\}$ . By abuse of notation we will refer to the vertex  $(s, 0) \in V \times \{0, 1, \dots, H\}$  as  $s$ , and we will refer to the vertex  $(r, H)$  as  $r$ . The graph  $\tilde{G}$  is obtained from  $G \times \{0, 1, \dots, H\}$  by deleting every vertex and edge which does not lie on a path from  $s$  to  $r$ . For every vertex  $v$  in  $\tilde{G}$ ,  $h(v)$  denotes the height of  $v$ , i.e. the number of edges on any path from  $v$  to  $r$ . Let  $d(v)$  denote the outdegree of  $v$ , and  $\Delta$  the maximum outdegree in  $G$ .

```

 $\varepsilon \leftarrow (\tilde{m} \log(\Delta) \log(\tilde{m}T)/T)^{1/3};$ 
 $\delta \leftarrow (\tilde{m} \log(\Delta) \log(\tilde{m}T)/T)^{1/3};$ 
 $\tau \leftarrow \lceil 2\tilde{m} \log(\tilde{m}T)/\delta \rceil;$ 
Initialize BEX( $v$ ) with parameter  $\varepsilon$  at each  $v \in \tilde{V}$ .
for  $\phi = 1, \dots, \lceil T/\tau \rceil$ , do
  for  $j = \tau(\phi-1)+1, \tau(\phi-1)+2, \dots, \tau\phi$  do /* Phase  $\phi$  */
    /* Sample a path  $\pi_j$  from  $s$  to  $r$ . */
    With probability  $\delta$ , /* Exploration */
      Choose  $e = (v, w)$  uniformly at random from  $\tilde{E}$ ;
      Construct  $\pi_j$  by joining random samples from
        prefix( $v$ ), suffix( $w$ ) using  $e$ ;
       $\pi_j^- \leftarrow \text{prefix}(v)$ ;  $\pi_j^0 \leftarrow \{e\}$ ;  $\pi_j^+ \leftarrow \text{suffix}(w)$ .
    Else, /* Exploitation */
      Sample  $\pi_j$  from suffix( $s$ );
       $\pi_j^- \leftarrow \emptyset$ ;  $\pi_j^0 \leftarrow \emptyset$ ;  $\pi_j^+ \leftarrow \pi_j$ .
    Receive feedback  $c_j(\pi_j)$ .
     $\chi_j(e) \leftarrow 1$  for all  $e \in \pi_j^0 \cup \pi_j^+$ .
  end /* End phase  $\phi$  */
 $\forall e \in \tilde{E}$ ,
   $\mu_\phi(e) \leftarrow \mathbf{E}[\sum_{j \in \phi} \chi_j(e)]$ 
   $\tilde{c}_\phi(e) \leftarrow (\sum_{j \in \phi} \chi_j(e) c_j(\pi_j)) / \mu_\phi(e)$ 
 $\forall v \in \tilde{V}$ , update BEX( $v$ ) using scores  $\tilde{c}_\phi(e)$ .
end /* End main loop */

```

Figure 3: Algorithm for online shortest paths

The algorithm (presented in Figure 3) requires three subroutines, described below:

- A “black-box expert algorithm” BEX( $v$ ) for each vertex  $v$ , which provides a probability distribution on outgoing edges from  $v$ .
- A sampling algorithm suffix( $v$ ) for sampling a random path from  $v$  to  $r$ .
- A sampling algorithm prefix( $v$ ) for sampling a random path from  $s$  to  $v$ .

Informally, BEX( $v$ ) is responsible for selecting an outgoing edge  $e = (v, w)$  lying on a path suffix( $v$ ) from  $v$  to  $r$  which is nearly as cheap (on average) as possible. Assuming that all vertices downstream from  $v$  are already selecting a nearly-cheapest path to  $r$ , the task of BEX( $v$ ) is therefore to identify an edge  $e$  so that the observed total cost of all edges from  $v$  to  $r$ , averaged over all sample paths traversing  $e$ , is nearly minimized. However, the feedback for each such sample path is the total *end-to-end* cost of the path, including the cost of edges lying between  $s$  and  $v$ , so it is necessary to cancel out the

“noise” contributed by such edges. This necessitates sampling this initial portion of the path from a rather complicated distribution  $\text{prefix}(v)$  which is described in Section 3.5.

To ensure that each  $\text{BEX}(v)$  receives enough feedback, the algorithm runs in phases of length  $\tau = \lceil 2\tilde{m} \log(\tilde{m}T)/\delta \rceil$ , with each phase simulating one round in the best-expert problem for  $\text{BEX}(v)$ . At each time step  $j$  within a phase  $\phi$ , a path  $\pi_j$  from  $s$  to  $r$  is sampled at random, independently, according to a rule which mixes “exploration” steps with probability  $\delta$  and “exploitation” steps with probability  $1 - \delta$ . In an exploration step, an edge  $e = (v, w)$  is selected at random, and the algorithm samples a random path through  $e$  using  $\text{prefix}(v), \text{suffix}(w)$ . This is illustrated in Figure 4. In an exploitation step, a path is selected according to the distribution  $\text{suffix}(s)$ , which simply uses the best-expert black box at each visited vertex to choose the next outgoing edge. In each step, the edges belonging to the prefix portion of the path are considered “tainted” and all other edges are marked. The marked edges receive a feedback score equal to the total cost of the sampled path. These scores are used to compute a cost vector  $\tilde{c}_\phi$  which is fed into  $\text{BEX}(v)$  at the end of the phase, so that the probability distribution on paths may be updated in the next phase. The formula defining  $\tilde{c}_\phi$  has a relatively simple interpretation: it is the total cost of all non-tainted samples on an edge, divided by the expected number of such samples. The tainted samples for an edge  $e$  are ignored because the portions of the path preceding and following  $e$  come from a conditional probability distribution which we cannot control, and could bias the score assigned to that edge in the case of an adaptive adversary.

**Theorem 3.1.** *The algorithm in Figure 3 achieves regret of*

$$O\left(H^2(mH \log \Delta \log(mHT))^{1/3} T^{2/3}\right)$$

*against an adaptive adversary, for paths of length  $\leq H$  in a graph with  $m$  edges and max-degree  $\Delta$ , during time  $T$ .*

Before proving this theorem, we must of course finish specifying the algorithm by specifying the implementations of  $\text{BEX}(v)$ ,  $\text{suffix}(v)$ ,  $\text{prefix}(v)$ .

### 3.3 Specification of $\text{BEX}(v)$

The implementation of  $\text{BEX}(v)$  relies on an algorithm which we may call the “best-expert algorithm with parameter  $\epsilon$ .” This algorithm is initialized with a parameter  $\epsilon$  and a finite set of  $K$  experts, which we will identify with  $\{1, 2, \dots, K\}$  henceforth. It receives as input a sequence of non-negative cost vectors  $c_1, \dots, c_t$ , with some known upper bound  $M$  on the costs; in our case, we’ll set this upper bound to be  $3H$ . It computes, for each  $j = 1, 2, \dots, t$  a probability distribution  $p_j$  on  $\{1, 2, \dots, K\}$ , depending only on  $c_1, \dots, c_{j-1}$ . If this black box is chosen to be one of the algorithms in [10, 13], then [10, 13] prove a tight upper bound on the regret of the algorithm. Specifically, for all  $k \in \{1, 2, \dots, K\}$ , the total cost of expert  $k$  is related to the algorithm’s expected cost by

$$\sum_{j=1}^t \sum_{i=1}^K p_j(i) c_j(i) \leq \sum_{j=1}^t c_j(k) + O\left(\epsilon t + \frac{\log K}{\epsilon}\right) M. \quad (5)$$

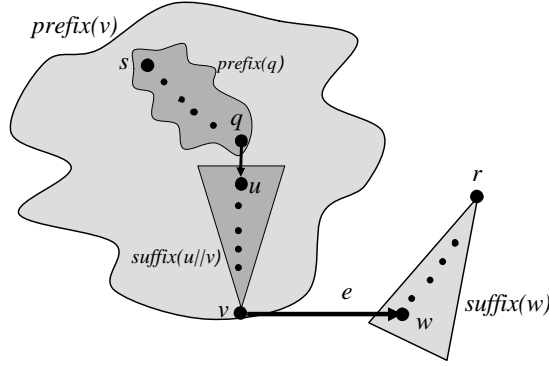


Figure 4: The recursive path sampling. Prefix distribution  $\text{prefix}(v)$  is generated recursively by concatenating  $\text{prefix}(q)$  for vertex  $q$  with  $\text{suffix}(u||v)$ , for a random edge  $(q, u)$  at prior layer.

### 3.4 Specification of $\text{suffix}(v)$

The probability distributions on outgoing edges, specified by the black-box expert algorithms  $\text{BEX}(v)$  at each vertex, naturally give rise to a probability distribution  $\text{suffix}(v)$  on paths from  $v$  to  $r$ . To sample a random path from  $\text{suffix}(v)$ , one chooses an outgoing edge from  $v$  according to the probability distribution returned by  $\text{BEX}(v)$ , traverses this edge to arrive at some vertex  $w$ , and continues sampling outgoing edges using the  $\text{BEX}$  black box at each vertex of the path until  $r$  is reached.

### 3.5 Specification of $\text{prefix}(v)$

We will define the distribution of path prefixes  $\text{prefix}(v)$  with the following goal in mind: in a step which contributes feedback for  $v$  (i.e. when  $v$  is incident to  $\pi_j^0 \cup \pi_j^+$ ), the portion of the path  $\pi_j$  preceding  $v$  should be distributed independently of  $v$ 's choice of outgoing edge. (For a precise mathematical statement, see Claim 3.2 below.) This property is desirable because the goal of  $\text{BEX}(v)$  is to learn to choose an edge  $e = (v, w)$  which approximately minimizes the average cost of  $e$  plus  $\text{suffix}(w)$ . However, the round-trip feedback scores observed by  $v$  contain an extra term accounting for the cost of the edges from  $s$  to  $v$ . The property proved in Claim 3.2 ensures that the expected value of this extra term does not depend on  $\text{BEX}(v)$ 's choice of an outgoing edge; hence it cannot bias  $\text{BEX}(v)$  against choosing the best outgoing edge.

The desired distribution  $\text{prefix}(v)$  is defined recursively, by induction on the distance from  $s$  to  $v$ , according to the rule of thumb that a random path drawn from  $\text{prefix}(v)$  should be indistinguishable from a random path, drawn according to the algorithm's sampling rule, in which  $v \in \pi^+$ . More precisely, if  $s = v$ , then  $\text{prefix}(v)$  is the empty path. Else, let  $\mathcal{F}_{<\phi}$  denote the  $\sigma$ -field generated by the algorithm's random choices prior to phase  $\phi$ , let

$$\begin{aligned} P_\phi(v) &= \Pr(v \in \pi_j^+ \mid \mathcal{F}_{<\phi}) \\ &= (1 - \delta) \Pr(v \in \text{suffix}(s) \mid \mathcal{F}_{<\phi}) + \sum_{e=(q,u) \in \tilde{E}} \left( \frac{\delta}{\tilde{m}} \right) \Pr(v \in \text{suffix}(u) \mid \mathcal{F}_{<\phi}), \end{aligned}$$

and let  $\text{suffix}(u||v)$  denote the distribution on  $u - v$  paths obtained by sampling a random path from  $\text{suffix}(u)$ , conditional on the event that the path passes through  $v$ , and taking the subpath beginning at  $u$  and ending at  $v$ . (A random sample from  $\text{suffix}(u||v)$  can be generated by a simple back-propagation algorithm. One uses dynamic programming to compute, for each edge  $e$ , the probability  $p_{u,e}$  that  $\text{suffix}(u)$  traverses  $e$ . Then, starting at  $v$ , one selects an incoming edge  $e$  with probability proportional to  $p_{u,e}$ , and the portion of the path preceding this edge is sampled recursively by the same procedure.) Now define  $\text{prefix}(v)$  to be the random path generated by the following rule:

- Sample from  $\text{suffix}(s||v)$  with probability

$$(1 - \delta) \Pr(v \in \text{suffix}(s) \parallel \mathcal{F}_{<\phi}) / P_\phi(v).$$

- For all  $e = (q, u) \in \tilde{E}$ , with probability

$$(\delta/\tilde{m}) \Pr(v \in \text{suffix}(u) \parallel \mathcal{F}_{<\phi}) / P_\phi(v),$$

sample from  $\text{suffix}(u||v)$ , prepend the edge  $e$ , and then prepend a random sample from  $\text{prefix}(q)$ .

**Claim 3.2.** *Conditional on  $\mathcal{F}_{<\phi}$  and the event  $v \in \pi_j$ , the sub-path of  $\pi_j$  beginning at  $s$  and ending at  $v$  is distributed independently of  $\chi_j(e)$  for all  $e \in \Delta(v), j \in \phi$ .*

*Proof.* Let  $\pi$  be any path from  $s$  to  $v$ . We will prove that

$$\Pr(\pi \subseteq \pi_j \parallel \chi_j(e) = 1 \wedge \mathcal{F}_{<\phi}) = \Pr(\text{prefix}(v) = \pi \parallel \mathcal{F}_{<\phi}).$$

This suffices to establish the claim, since the right side is manifestly independent of  $\chi_j(e)$ . For the remainder of the proof, we will simplify notation by dropping the “ $\mathcal{F}_{<\phi}$ ” from our expressions for probabilities; each such expression should implicitly be interpreted as a conditional probability, in which we condition on  $\mathcal{F}_{<\phi}$  in addition to whatever other events or random variables might be present in the expression.

If  $\chi_j(e) = 1$ , then either  $e \in \pi_j^0$  or  $e \in \pi_j^+$ . Now,

$$\Pr(\pi \subseteq \pi_j \parallel e \in \pi_j^0) = \Pr(\text{prefix}(v) = \pi);$$

this is merely a restatement of the procedure for sampling a random path through edge  $e$  in an exploration step. It remains to show that

$$\Pr(\pi \subseteq \pi_j \parallel e \in \pi_j^+) = \Pr(\text{prefix}(v) = \pi).$$

We first observe that, conditional on  $v$  belonging to  $\pi_j^+$ , the outgoing edge from  $v$  is sampled according to the black-box distribution at  $v$ , independently of the path preceding  $v$ ; thus

$$\Pr(\pi \subseteq \pi_j \parallel e \in \pi_j^+) = \Pr(\pi \subseteq \pi_j \parallel v \in \pi_j^+).$$

Now,

$$\begin{aligned} P_\phi(v) \Pr(\pi \subseteq \pi_j \parallel v \in \pi_j^+) &= \Pr(v \in \pi_j^+) \Pr(\pi \subseteq \pi_j \parallel v \in \pi_j^+) \\ &= \Pr(\pi \subseteq \pi_j) \\ &= (1 - \delta) \Pr(\pi \subseteq \pi_j \parallel \pi_j^0 = \emptyset) + \sum_{e=(u,w) \in \tilde{E}} \left( \frac{\delta}{\tilde{m}} \right) \Pr(\pi \subseteq \pi_j \parallel \pi_j^0 = \{e\}) \\ &= P_\phi(v) \Pr(\pi = \text{prefix}(v)), \end{aligned}$$

where the last line follows from the construction of the distribution  $\text{prefix}(v)$  specified above. Dividing both sides by  $P_\phi(v)$ , we obtain  $\Pr(\pi \subseteq \pi_j \mid v \in \pi_j^+) = \Pr(\pi = \text{prefix}(v))$ , as desired.  $\square$

### 3.6 Analysis of the algorithm

In this section we'll prove Theorem 3.1, which bounds the algorithm's regret. Let  $t := \lceil T/\tau \rceil$  denote the number of phases. Let  $c^-(v), c^+(v)$  be the average costs of the paths  $\text{prefix}(v), \text{suffix}(v)$ , respectively, i.e.

$$\begin{aligned} c^-(v) &= \frac{1}{T} \sum_{j=1}^T E[c_j(\text{prefix}(v))] \\ c^+(v) &= \frac{1}{T} \sum_{j=1}^T E[c_j(\text{suffix}(v))]. \end{aligned}$$

Let  $OPT(v)$  denote the average cost of the best fixed path from  $v$  to  $r$ , i.e.

$$OPT(v) = \min_{\text{paths } \pi: v \rightarrow r} \frac{1}{T} \sum_{j=1}^T \mathbf{E}[c_j(\pi)],$$

where the expectation is over the algorithm's random choices, which in turn influence the cost functions  $c_j$  because the adversary is adaptive.

In the case of an oblivious adversary,  $OPT(v)$  is simply the cost of the best path from  $v$  to  $r$ . In the case of an adaptive adversary, it is a bit tougher to interpret  $OPT(v)$ : the natural definition would be " $OPT(v)$  is the *expectation of the minimum cost* of a fixed path from  $v$  to  $r$ ," but instead we have defined it as the *minimum of the expected cost* of a fixed path, by analogy with the corresponding definition in [2]. We leave open the question of whether a similar regret bound can be established relative to the more natural definition of  $OPT$ .

Our plan is to bound  $c^+(v) - OPT(v)$  by induction on  $h(v)$ . We think of this bound as a "local performance guarantee" at the vertex  $v$ . The local performance guarantee at  $s$  supplies a bound on the expected regret of the algorithm's exploitation steps; we'll combine this with a trivial bound on the expected cost of the exploration steps to obtain a global performance guarantee which bounds the expected regret over all time steps.

#### 3.6.1 Local performance guarantees

Fix a vertex  $v$  of degree  $d$ , and let  $p_\phi$  denote the probability distribution on outgoing edges supplied by the black-box expert algorithm at  $v$  during phase  $\phi$ . The stated performance guarantee for the best-expert algorithm (5) ensures that for each edge  $e_0 = (v, w_0)$ ,

$$\sum_{\phi=1}^t \sum_{e \in \Delta(v)} p_\phi(e) \tilde{c}_\phi(e) \leq \sum_{\phi=1}^t \tilde{c}_\phi(e_0) + O\left(\epsilon H t + \frac{H \log \Delta}{\epsilon}\right),$$

provided  $M = \max\{\tilde{c}_\phi(e) : 1 \leq \phi \leq t, e \in \tilde{E}\} \leq 3H$ . By Chernoff bounds, the probability that  $\tilde{c}_\phi(e) > 3H$  is less than  $(\tilde{m}T)^{-2}$ , because  $\tilde{c}_\phi(e)$  can only exceed  $3H$  if the number of samples of  $e$  in

phase  $\phi$  exceeds its expectation by a factor of 3, and the expected number of such samples is at least  $2 \log(\tilde{m}T)$ . Applying the union bound, we see that the probability that  $M > 3H$  is less than  $(\tilde{m}T)^{-1}$ . We'll subsequently ignore this low-probability event, since it can contribute at most  $(HT)/(\tilde{m}T) \leq 1$  to the overall expected regret.

Expanding out the  $\tilde{c}_\phi(\cdot)$  terms above, using the definition of  $\tilde{c}_\phi$ , we get

$$\sum_{\phi=1}^t \sum_{e \in \Delta(v)} \sum_{j \in \phi} \frac{p_\phi(e)}{\mu_\phi(e)} \chi_j(e) c_j(\pi_j) \leq \sum_{\phi=1}^t \sum_{j \in \phi} \frac{1}{\mu_\phi(e_0)} \chi_j(e_0) c_j(\pi_j) + O\left(\epsilon H t + \frac{H \log \Delta}{\epsilon}\right). \quad (6)$$

Now let's take the expectation of both sides with respect to the algorithm's random choices. We'll use the following fact.

**Claim 3.3.** *If  $e = (v, w)$  then*

$$\mathbf{E}_\phi[\chi_j(e) c_j(\pi_j)] = \left(\frac{\mu_\phi(e)}{\tau}\right) (A_j(v) + B_j(w) + \mathbf{E}_\phi[c_j(e)]),$$

where  $\mathbf{E}_\phi[\cdot]$  denotes  $\mathbf{E}[\cdot | \mathcal{F}_{<\phi}]$ , and

$$\begin{aligned} A_j(v) &= \mathbf{E}_\phi[c_j(\text{prefix}(v))] \\ B_j(w) &= \mathbf{E}_\phi[c_j(\text{suffix}(w))]. \end{aligned}$$

*Proof.* Conditional on  $\mathcal{F}_{<\phi}$  and on the event  $\chi_j(e) = 1$ , the portion of the path preceding  $v$  is distributed according to  $\text{prefix}(v)$  (this was proved in claim 3.2) and the portion of the path following  $w$  is distributed according to  $\text{suffix}(w)$  (this follows from the definition of  $\chi_j(e)$  and of  $\text{suffix}(w)$ ). Moreover, for any edge  $e'$ ,

$$\mathbf{E}_\phi[c_j(e') \mid \chi_j(e) = 1] = \mathbf{E}_\phi[c_j(e')],$$

because  $\chi_j(e)$  is independent of any decisions made by the algorithm during phase  $\phi$  and before step  $j$ , while the adversary's choice of  $c_j$  depends only on  $\mathcal{F}_{<\phi}$  and on the decisions made in phase  $\phi$  before step  $j$ . Thus,

$$\mathbf{E}_\phi[c_j(\pi_j) \mid \chi_j(e) = 1] = A_j(v) + B_j(w) + \mathbf{E}_\phi[c_j(e)]$$

$$\begin{aligned} \mathbf{E}_\phi[\chi_j(e) c_j(\pi_j)] &= \Pr(\chi_j(e) = 1 \mid \mathcal{F}_{<\phi}) \mathbf{E}_\phi[c_j(\pi_j) \mid \chi_j(e) = 1] \\ &= \left(\frac{\mu_\phi(e)}{\tau}\right) (A_j(v) + B_j(w) + \mathbf{E}_\phi[c_j(e)]). \end{aligned}$$

□

Now consider taking the expectation of both sides of (6). The left side will become

$$\begin{aligned} &\sum_{\phi=1}^t \sum_{e \in \Delta(v)} \sum_{j \in \phi} \frac{p_\phi(e)}{\mu_\phi(e)} \cdot \frac{\mu_\phi(e)}{\tau} \cdot (A_j(v) + B_j(w) + \mathbf{E}_\phi[c_j(e)]) \\ &= \frac{1}{\tau} \sum_{j=1}^T \sum_{e \in \Delta(v)} p_\phi(e) (A_j(v) + B_j(w) + \mathbf{E}_\phi[c_j(e)]), \end{aligned}$$

while the sum on the right side will become

$$\sum_{\phi=1}^t \sum_{j \in \phi} \frac{1}{\mu_{\phi}(e_0)} \cdot \frac{\mu_{\phi}(e_0)}{\tau} \cdot (A_j(v) + B_j(w_0) + \mathbf{E}_{\phi}[c_j(e_0)]) = \frac{1}{\tau} \sum_{j=1}^T (A_j(v) + B_j(w_0) + \mathbf{E}_{\phi}[c_j(e_0)]).$$

Plugging this back into (6), the terms involving  $\text{prefix}(v)$  on the left and right sides will cancel, leaving us with

$$\frac{1}{\tau} \sum_{j=1}^T \sum_{e \in \Delta(v)} p_{\phi}(e) (\mathbf{E}_{\phi}[c_j(e)] + B_j(w)) \leq \frac{1}{\tau} \sum_{j=1}^T (\mathbf{E}_{\phi}[c_j(e_0)] + B_j(w_0)) + O\left(\epsilon Ht + \frac{H \log \Delta}{\epsilon}\right).$$

Note that the left side is equal to

$$\frac{1}{\tau} \sum_{j=1}^T \mathbf{E}[c_j(\text{suffix}(v))] = c^+(v)/\tau,$$

while the right side is equal to

$$\frac{1}{\tau} \left( \sum_{j=1}^T \mathbf{E}[c_j(e_0)] \right) + c^+(w_0)/\tau + O\left(\epsilon Ht + \frac{H \log \Delta}{\epsilon}\right).$$

Thus we have derived

$$c^+(v) \leq c^+(w_0) + \sum_{j=1}^T \mathbf{E}[c_j(e_0)] + O\left(\epsilon HT + \frac{\tau H \log \Delta}{\epsilon}\right). \quad (7)$$

### 3.6.2 Global performance guarantee

**Claim 3.4.** *Let  $\Delta$  denote the maximum outdegree in  $G$ . For all  $v$ ,*

$$c^+(v) \leq OPT(v) + O\left(\epsilon HT + \frac{\tau H \log \Delta}{\epsilon}\right) h(v).$$

*Proof.* The proof uses the following simple observation about  $OPT(v)$ :

$$OPT(v) = \min_{e_0=(v,w_0)} \left\{ \sum_{j=1}^T \mathbf{E}[c_j(e_0)] + OPT(w_0) \right\}.$$

Now the claim follows easily from equation (7) by induction on  $h(v)$ . □

**Theorem 3.5.** *The algorithm in Figure 3 suffers regret  $O\left(H^{7/3}(m \log(\Delta) \log(mHT))^{1/3} T^{2/3}\right)$ .*

*Proof.* The exploration steps are a  $\delta$  fraction of all time steps, and each contributes at most  $H$  to the regret, so they contribute at most  $\delta TH$  to the regret. The contribution of the exploitation steps to the regret is at most  $c^+(s) - OPT(s)$ . Applying Claim 3.4 above, and substituting  $\tau = \frac{2\tilde{m} \log(\tilde{m}T)}{\delta}$ , we see that

$$c^+(s) - OPT(s) = O\left(\epsilon T + \frac{2\tilde{m} \log(\Delta) \log(\tilde{m}T)}{\epsilon\delta}\right) H^2.$$

Thus

$$\begin{aligned} \text{Regret} &\leq \delta TH + O\left(\epsilon T + \frac{2\tilde{m} \log(\Delta) \log(\tilde{m}T)}{\epsilon\delta}\right) H^2 \\ &\leq O\left(\delta T + \epsilon T + \frac{2\tilde{m} \log(\Delta) \log(\tilde{m}T)}{\epsilon\delta}\right) H^2. \end{aligned}$$

Plugging in the parameter settings specified in the theorem, we obtain the desired conclusion.  $\square$

## 4 Acknowledgements

We would like to thank Avrim Blum, Adam Kalai, Yishay Mansour, and Santosh Vempala for helpful discussions relating to this work, and Brendan McMahan for pointing out an error in a preliminary version of this paper.

## References

- [1] R. Agrawal. The continuum-armed bandit problem. *SIAM J. Control and Optimization*, 33:1926–1951, 1995.
- [2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.
- [3] B. Awerbuch, D. Holmer, H. Rubens, and R. Kleinberg. Provably competitive adaptive routing. In *Proceedings of the 31st IEEE INFOCOM*, 2005, to appear.
- [4] B. Awerbuch and R. Kleinberg. Adaptive routing with end-to-end feedback: distributed learning and geometric approaches. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 45–53. Chicago, 2004.
- [5] B. Awerbuch and Y. Mansour. Adapting to a reliable network path. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing*, pages 360–367. Boston, 2003.
- [6] J. Barnett and N. Srebro, personal communication, February 2005.
- [7] D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. London: Chapman and Hall, 1985.

- [8] A. Flaxman, A. Kalai, and H. B. McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 385–394. Vancouver, 2005.
- [9] J.C. Gittins and D.M. Jones. A dynamic allocation index for the sequential design of experiments. In J. Gani, K. Sakadi, and I. Vinczo, eds. *Progress in Statistics*, pages 241-266. Amsterdam: North-Holland, 1974.
- [10] A. Kalai and S. Vempala. Efficient algorithms for the online decision problem. In *Proc. of 16th Conference on Computational Learning Theory*. Washington, DC, 2003.
- [11] R. Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *Advances in Neural Information Processing Systems 17*, 2004, to appear.
- [12] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocations rules. *Adv. in Appl. Math.* 6:4-22, 1985.
- [13] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–260, 1994.
- [14] H. B. McMahan and A. Blum. Online geometric optimization in the bandit setting against an adaptive adversary. In *Proceedings of the 17th Conference on Learning Theory*. Banff, 2004.
- [15] E. Takimoto and M. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research* 4 (2003): 773–818.
- [16] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, pages 928–936, 2003.