Adaptive Routing with End-to-End feedback: Distributed Learning and Geometric Approaches

Baruch Awerbuch * Robert D. Kleinberg †

ABSTRACT

Minimal delay routing is a fundamental task in networks. Since delays depend on the (potentially unpredictable) traffic distribution, online delay optimization can be quite challenging. While uncertainty about the current network delays may make the current routing choices sub-optimal, the algorithm can nevertheless try to learn the traffic patterns and keep adapting its choice of routing paths so as to perform nearly as well as the best static path.

This online shortest path problem is a special case of *online linear optimization*, a problem in which an online algorithm must choose, in each round, a strategy from some compact set $\mathcal{S} \subseteq \mathbb{R}^d$ so as to try to minimize a linear cost function which is only revealed at the end of the round. Kalai and Vempala [4] gave an algorithm for such problems in the *transparent feedback* model, where the entire cost function is revealed at the end of the round. Here we present an algorithm for online linear optimization in the more challenging *opaque feedback* model, in which only the cost of the chosen strategy is revealed at the end of the round. In the special case of shortest paths, opaque feedback corresponds to the notion that in each round the algorithm learns only the end-to-end cost of the chosen path, not the cost of every edge in the network.

We also present a second algorithm for online shortest paths, which solves the shortest-path 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 better than the first, as measured by their additive regret.

*Department of Computer Science, Johns Hopkins University, 3400 N. Charles Street, Baltimore MD 21218, USA. Email: 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. Supported by a Fannie and John Hertz Foundation Fellowship.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*

General Terms

Algorithms, Theory

Keywords

Routing, online decision problem, multi-armed bandit problem, online linear optimization

1. INTRODUCTION

1.1 The learning models

We consider online learning algorithms (or equivalently, repeated games between an online algorithm and an adversary) in a framework which is typically dubbed "the multi-armed bandit problem." The name derives from the metaphor of a gambler selecting slot machines in a rigged casino [1], though Internet routing is the most compelling motivation in the present context. The general framework, adopted from [1], can be described as follows.

- A set $\mathcal{S}$ of strategies is given.
- The algorithm interacts with an adversary in a series of T steps.
- In each step j, the algorithm picks a strategy $x_j \in \mathcal{S}$, and the adversary selects a cost function $c_j : \mathcal{S} \to \mathbb{R}$. The adversary is *adaptive*, in that c_j may depend on $\{x_i : i < j\}$.
- The algorithm incurs cost $c_j(x_j)$, and receives as feedback the value of $c_j(x_j)$.
- The algorithm's regret is defined as the difference in expected cost between the algorithm's sequence of choices and that of the best fixed strategy in $\mathcal{S}$, i.e.,

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

The following two variations of the model are of interest.

Transparent feedback means that the entire cost function c_j is revealed, in contrast to the *opaque* feedback model specified above, in which only the cost of the chosen strategy

is revealed. In the learning literature, the transparent and opaque models are referred to as the "best expert" [7] and "multi-armed bandit" problems [1], respectively.

An **oblivious adversary** (in contrast to an adaptive adversary) does not see the algorithm's past decisions. It chooses the entire sequence of cost functions $c_1, \ldots, c_T$ up front.

1.2 The problems

We focus in this paper on two problems, both in the opaque feedback model above.

- *Dynamic shortest paths:* The strategy space consists of paths from a sender s to a receiver r in a directed graph $G = (V, E)$. The cost function is specified by assigning lengths to the edges of G, charging a path for the sum of its edge lengths, and receiving the total path length as feedback. The goal is to match the cost of a shortest path of at most H edges from s to r, in the aggregate cost metric.
- *Online linear optimization:* In this more general problem [4], the strategy space is a compact subset of $\mathbb{R}^d$, the cost functions are linear, and the feedback consists of the cost of the chosen point.

The dynamic shortest paths problem may be applied to overlay network routing, by interpreting edge costs as link delays. In our formulation, the feedback 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.

1.3 New contributions

We present:

1. The first *opaque-feedback* online linear optimization algorithm, against an oblivious adversary, with regret $O(T^{2/3}Md^{5/3})$, where d is the dimension of the problem space and M is an upper bound on the costs in each step.
2. The first *polynomial regret* online shortest-path algorithm against an adaptive adversary, achieving regret $O\left(T^{2/3}H^{7/3}(m \log \Delta \log(m \cdot T))^{1/3}\right)$ for the shortest paths problem of length H in a graph with m edges and max-degree Δ, during T time steps.

The online linear optimization algorithm may be applied in the special case of shortest paths of length H; 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 Algorithm 2, and in a weaker adversarial model. As in [1], 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 [4, 6, 8].

For the linear optimization problem, the novel idea in our work is to compute a special basis of the linear strategy space, called a *barycentric spanner*, which 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, obtaining "good enough" estimates about the costs of all other strategies, and feeding these estimates into the linear optimization algorithm of Kalai and Vempala [4], which accomplishes small regret and polynomial computation time with the help of an optimization oracle. We also provide a polynomial-time construction of the barycentric spanner, using the same optimization oracle as a subroutine. However, our analysis of the algorithm must assume an oblivious adversary, as an adaptive adversary may be able to outwit this strategy by choosing the cost functions carefully.

Our shortest paths algorithm, which works against an adaptive adversary, can be viewed as a distributed learning algorithm with a "local expert" at each node in the graph making local decisions about which outgoing edge to take on the shortest path towards the receiver r. Ideally, each node would know the past costs of each edge. However, this information is not available; instead the local expert is only given the total end-to-end path cost, for all prior sample paths passing through that vertex. The local agent 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 algorithm, and it is somewhat surprising that there exists an online algorithm which successfully defeats such an adversary.

1.4 Comparison with existing work

The only existing solution for either one of the above problems, in the *opaque* model considered here, has been given by Auer et al [1], who presented an algorithm achieving $O(\sqrt{TK \log K})$ regret for the K-armed bandit problem. Note that the for the shortest paths problem, the cardinality of the strategy space may be exponential in the size n of the vertex set of 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 [1] do in fact exhibit exponential ($2^{\Omega(n)}$) regret. All other existing work does not apply to our case as it assumes much more favorable feedback models than the ones considered in this paper.

Below we describe relevant work for other feedback models. For the full-feedback (transparent) case, many results are known. Littlestone and Warmuth's seminal weighted-majority algorithm [7] achieves $O(\sqrt{T \log K})$ regret for K strategies and cost functions taking values in $[0, 1]$, a case known as the *best-expert problem* because of the metaphor of learning from the advice of K experts. Kalai and Vempala [4] considered the best-expert problem when the strategy set $\mathcal{S}$ is contained in $\mathbb{R}^d$ and the cost functions are linear (and again take values in $[0, 1]$), presenting an algorithm which achieves $O(\sqrt{T \log d})$ regret in this case. (This bound depends only on the dimension d, hence it is applicable even if the cardinality of $\mathcal{S}$ is exponential or infinite.) By considering the set of $s-r$ paths in G as the vertices of the polytope of unit flows from s to r, Kalai and Vempala demonstrated that $O(\sqrt{T \log m})$ regret is achievable in the setting where the lengths of all edges are revealed at the end of each time

step, i.e. the "best-expert version" of the dynamic shortest path problem.

In subsequent work, Awerbuch and Mansour [2] used a "prefix" feedback model, where feedback was available on all the prefixes of the path selected, and showed that learning the shortest path is possible by using the best-expert algorithm [7] as a black box. Blum et al [3] have considered a "partially transparent model" where feedback is provided on *each edge* of the selected path, but not on the other edges of the graph.

Our Algorithm 2 is structurally similar to the Awerbuch-Mansour algorithm [2] in that it proceeds in phases and involves local decision making at the nodes. However, [2] requires richer feedback, and in addition it works only for an oblivious adversary. The stronger *adaptive* adversary model as in [1] is capable of biasing the Awerbuch-Mansour algorithm's estimates by a careful choice of the cost functions in each phase, causing the algorithm to fail. 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 obtain a regret bound against an adaptive adversary. If the adversary is oblivious, it would be possible to use any *stationary* prefix distribution, as in [2].

2. ONLINE LINEAR OPTIMIZATION

2.1 Problem formulation

As in [4], we consider an online optimization problem in which the strategy space is a compact set $S \subseteq \mathbb{R}^d$, and the cost functions are linear functions specified by a sequence of cost vectors $c_1, \ldots, c_T \in \mathbb{R}^d$ chosen by an *oblivious* adversary.

On each time step, the adversary picks a cost vector c_j and the algorithm chooses a strategy $x_j \in S$, receiving as feedback the value of $c_j \cdot x_j$, i.e. the cost of the chosen strategy. The cost vectors themselves are never revealed; it is this feature which distinguishes our problem from the one solved in [4]. We will assume that there is an upper bound M on the costs, i.e. that $|c_j \cdot x| \leq M$ for all $x \in S$.

Note that this problem generalizes the classical multi-armed bandit problem as well as the dynamic shortest path problem stated above. The multi-armed bandit problem is the case where S is a set of d points composing a basis of $\mathbb{R}^d$. To interpret the online shortest-paths problem in this framework, the vector space in question is $\mathbb{R}^E$, the space of real-valued functions on the edge set. A path may be considered as a function taking the value 1 on each of its edges, 0 elsewhere. The strategy space S is again a finite point set, consisting of all the points in $\mathbb{R}^E$ represented by paths from s to r with length $\leq H$. The cost functions are linear; their coefficients are in fact given by the corresponding edge lengths.

2.2 Overview of algorithm

As stated in the introduction, the transparent-feedback version of the problem has been solved by Kalai and Vempala in [4]. Our plan is to 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 fraction of steps

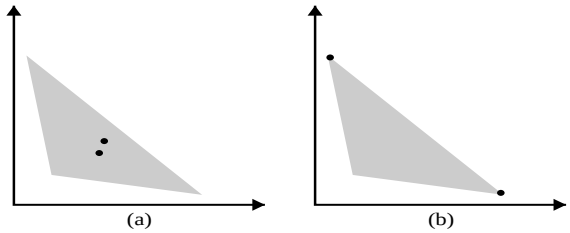

Figure 1: (a) A bad sampling set (b) A barycentric spanner.

which are used for explicitly sampling the costs of certain strategies, and a much larger fraction of "exploitation" steps in which we choose our strategy according to the black box, with the aim of minimizing cost. The feedback to the black box at the end of the 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 vector. This shortcoming of the algorithm explains why our analysis works only in 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 [1], 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 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 2.2. 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 2.3; 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.

2.3 Barycentric spanners

Definition 2.1. Let $S \subseteq \mathbb{R}^d$ be a subset which is not contained in any lower-dimensional linear subspace of $\mathbb{R}^d$. A point set $X = \{x_1, \ldots, x_d\} \subseteq S$ 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 $\mathbb{R}^d$ which is not contained in any linear subspace of dimension less than d, then S has a barycentric spanner.*

Proof. Choose a subset $X = \{x_1, \ldots, x_d\} \subseteq S$ maximizing $|\det(x_1, \ldots, x_d)|$. (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

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

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

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

Observation 2.3. The proof of Proposition 2.2 actually established the following stronger fact. Let $X = \{x_1, \ldots, x_d\}$ be a subset of S with the property that for any x in S and any $i \in \{1, \ldots, d\}$,

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

Then X is a C-approximate barycentric spanner for S. Here X_{-i} denotes the following $(d-1)$-tuple of vectors

$$X_{-i} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d).$$

Proposition 2.4. *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 $(\mathbf{e_1}, \ldots, \mathbf{e_d})$ 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 $(x_1, x_2, \ldots, x_d)$ contained in S, by replacing the original basis vectors $(\mathbf{e_1}, \ldots, \mathbf{e_d})$ 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 ℓ_i evaluates to zero on X_{-i}, and is nonzero on x_0. (ℓ_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 R^d contained in S. */
(x_1, ..., x_d) ← (e_1, ..., e_d);
for i = 1, 2, ..., d do
    /* Replace x_i with an element of S,
       while keeping it linearly independent from X_{-i}. */
    x_i ← arg max_{x∈S} |det(x, X_{-i})|;
end
/* Transform basis into approximate barycentric spanner. */
while ∃x ∈ S, i ∈ {1, ..., d} satisfying
          |det(x, X_{-i})| > C|det(x_i, X_{-i})|
    x_i ← x;
end
return (x_1, x_2, ..., x_d)
```

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

Lemma 2.5 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 $i \in \{1, \ldots, d\}$. At termination, $(x_1, \ldots, x_d)$ is a 2-approximate barycentric spanner, by Observation 2.3. □

Lemma 2.5. *The total number of iterations of the "while" loop is $O(d \log_C(d))$.*

Proof. Let $M_i = (x_1, x_2, \ldots, x_i, \mathbf{e_{i+1}}, \ldots, \mathbf{e_d})$ be the matrix whose columns are the basis vectors at the end of the i-th iteration of the 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, $(x_1, \ldots, x_d)$ 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 $u_i := \mathbf{e_i}^\mathsf{T} M_i^{-1}$, 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

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

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 $x_i = \arg\max_{x \in S} |\ell_i(x)|$. In particular each column of UM' is a vector in $[-1, 1]^d$ and therefore has length $\leq d^{1/2}$. Hence $\det(UM') \leq d^{d/2}$. (The determinant of a matrix cannot exceed the product of the L^2-norms of its column vectors.) 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 $\det(UM) = 1$. □

2.4 Algorithm

Without loss of generality, assume that S is not contained in any proper linear subspace of $\mathbb{R}^d$; otherwise, we may re-

place $\mathbb{R}^d$ with its linear subspace span(S) and run the algorithm in this subspace.

The algorithm will employ a subroutine known as the "Kalai-Vempala algorithm with parameter ε." (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 $c_j : S \to \mathbb{R}$, $(1 \le j \le t)$, 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 if $x^{(1)}, \ldots, x^{(t)}$ are random samples from $p_1, \ldots, p_t$, respectively, and x is any point in S,

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

See [4] for a description and analysis of the Kalai-Vempala algorithm with parameter ε. Their paper differs from ours in that they assume the cost vectors satisfy $\|c_j\|_1 \le 1$, and express the regret bound as

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

where D is the L^1-diameter of S. To derive (2) from this, let $\{x_1, \ldots, x_d\}$ be a 2-approximate barycentric spanner for S, and transform the coordinate system by mapping x_i to $(Md)\mathbf{e_i}$, for $i = 1, \ldots, d$. This maps S to a set whose L^1-diameter satisfies $D \le 4Md^2$, by the definition of a 2-approximate barycentric spanner. The cost vectors in the transformed coordinate system have no coordinate 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 ε, where ε is to be determined later. Divide the timeline $1, 2, \ldots, T$ into phases of length $\tau = \lceil d/\delta \rceil$, where δ is also a parameter to be determined later. (The time steps in phase ϕ are numbered $\tau(\phi-1)+1, \tau(\phi-1)+2, \ldots, \tau\phi$. Call this set of time steps $\mathcal{T}_\phi$.) Within each phase, select a subset of d time steps uniformly at random, and choose a random one-to-one correspondence between these time steps and the elements of X. The step in phase ϕ corresponding to $x_i \in X$ will be called the "sampling step for x_i in phase ϕ;" 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 chooses its strategy using the K-V black box.

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

Theorem 2.6. *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_ϕ satisfies $|c_\phi \cdot x_i| \le M$ for all $x_i \in X$, and that its expectation is $\bar{c}_\phi \stackrel{\text{def}}{=} \frac{1}{\tau}\sum_{j \in \mathcal{T}_\phi} c_j$.

Let $t = \lceil T/\tau \rceil$. The performance guarantee for the K-V algorithm ensures that for all $x \in S$,

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

where x_ϕ is a random sample from the probability distribution specified by the black box in phase ϕ. Henceforth we'll 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_ϕ and x_j are independent random variables: c_ϕ depends only on decisions made by the algorithm in phase ϕ, while x_j depends only on data fed to the K-V black box before phase ϕ, and random choices made by the K-V box during phase ϕ. 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 $x \in S$,

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

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

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

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

$$\textit{Regret} = O(T^{2/3} M d^{5/3}).$$

□

2.5 Application to dynamic shortest paths

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 $\le H$ 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 $\mathbb{R}^d$ by assigning to each flow a cost equal to 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 2.4, we obtain

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

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

For $j = 1, 2, ..., T$, an adversary selects a cost function $C_j : E \to [0, 1]$ for a directed graph $G(V, E)$. 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 (end-to-end delay). 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.

3.2 Algorithm Structure

As in [2], the algorithm will transform our arbitrary input graph $G(V, E)$ with n vertices and m edges into a levelled directed acyclic graph $\tilde{G} = (\tilde{V}, \tilde{E})$, such that a replica of each node is present at each layer of the graph. Each layer of the graph contains n vertices and m edges, and altogether there are $|\tilde{V}| = \tilde{n} = n \cdot H$ nodes and $|\tilde{E}| = \tilde{m} = m \cdot H$ edges in $\tilde{G}$. The top level of $\tilde{G}$ contains a source vertex s, and the bottom level contains a receiver vertex 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 Δ the maximum outdegree in G.

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

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

Informally, $\texttt{BEX}(v)$ is responsible for selecting an outgoing edge $e = (v, w)$ lying on a path $\text{suffix}(v)$ 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 $\texttt{BEX}(v)$ 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.

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

Figure 3: Algorithm for online shortest paths

To ensure that each $\texttt{BEX}(v)$ 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 $\texttt{BEX}(v)$. At each time step j within a phase ϕ, a path π_j from s to r is sampled at random, independently, according to a rule which mixes "exploration" steps with probability δ 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 $\texttt{BEX}(v)$ 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 Δ, during time T.

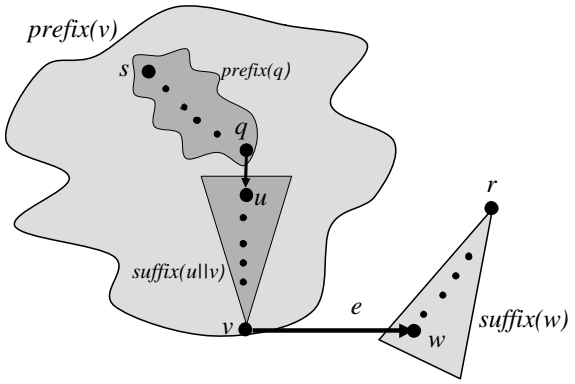

Figure 4: The recursive path sampling. Prefix distribution prefix(v) **is generated recursively by concatenating** prefix(q) **for vertex** q **with** suffix($u\|v$)**, for a random edge** (q, u) **at prior layer.**

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

3.3 Specification of BEX(v)

The implementation of BEX(v) relies on an algorithm which we may call the "best-expert algorithm with parameter ϵ." This algorithm is initialized with a parameter ϵ and a finite set of K experts, which we will identify with $\{1, 2, \ldots, K\}$ henceforth. It receives as input a sequence of non-negative cost vectors $c_1, \ldots, c_t$, 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 $j = 1, 2, \ldots, t$ a probability distribution p_j on $\{1, 2, \ldots, K\}$, depending only on $c_1, \ldots, c_{j-1}$. If this black box is chosen to be one of the algorithms in [4, 5], then [4, 5] prove a tight upper bound on the regret of the algorithm. Specifically, for all $k \in \{1, 2, \ldots, K\}$, the total cost of option k is related to the algorithm's expected cost by

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

3.4 Specification of suffix(v)

The probability distributions on outgoing edges, specified by the black-box expert algorithms BEX(v) at each vertex, naturally give rise to a probability distribution suffix(v) on paths from v to r. To sample a random path from suffix(v), choose an outgoing edge from v according to the probability distribution returned by BEX(v), traverse this edge to arrive at some vertex w, and continue sampling outgoing edges using the best-expert black box at each visited vertex until r is reached.

3.5 Specification of prefix(v)

We will define the distribution of path prefixes 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 π_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 BEX(v) is to learn to choose an edge $e = (v, w)$ which approximately minimizes the average cost of e plus 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 this extra term contributes an equal cost, in expectation, for all of v's outgoing edges; hence it cannot bias BEX(v) against choosing the best outgoing edge.

The desired distribution 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 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 prefix(v) is the empty path. Else, let $\mathcal{F}_{<\phi}$ denote the σ-field generated by the algorithm's random choices prior to phase ϕ, let

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

and let suffix($u\|v$) denote the distribution on $u - v$ paths obtained by sampling a random path from suffix(u), conditional on the event that the path passes through v, and taking the subpath reaching from u to v. (A random sample from suffix($u\|v$) can be generated in linear time by a simple back-propagation algorithm, in which the edge preceding v is selected at random from the requisite conditional distribution, and the portion of the path preceding this edge is sampled recursively by the same procedure.) Then prefix(v) is the random path generated by the following rule:

- Sample from suffix($s\|v$) with probability
$$(1-\delta) \Pr(v \in \text{suffix}(s) \| \mathcal{F}_{<\phi}) / P_\phi(v).$$
- For all $e = (q, u) \in E$, with probability
$$(\delta/\tilde{m}) \Pr(v \in \text{suffix}(u) \| \mathcal{F}_{<\phi}) / P_\phi(v),$$
sample from suffix($u\|v$), prepend the edge e, and then prepend a random sample from prefix(q).

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

Proof. Let π be any path from s to v. We will prove that

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

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,

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

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

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

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

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

Now,

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

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

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 prefix(v), 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.

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

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, adopting the corresponding definition in [1]. 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_ϕ denote the probability distribution on outgoing edges supplied by the black-box expert algorithm at v during phase ϕ. The stated performance guarantee for the best-expert algorithm (5) ensures that for each edge $e_0 = (v, w_0)$,

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

provided $M = \max\{\tilde{C}_\phi(e) : 1 \le \phi \le t, e \in \tilde{E}\} \le 3H$. 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 ϕ exceeds its expectation by a factor of 3, and the expected number of samples is $\ge 2\log(\tilde{m}T)$. 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 $(HT)/(\tilde{m}T) \le 1$ to the overall expected regret.

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

$$\begin{aligned} & \sum_{\phi=1}^{t} \sum_{e\in\Delta(v)} \sum_{j\in\phi} \frac{p_\phi(e)}{\mu_\phi(e)} \chi_j(e) C_j(\pi_j) \qquad (6) \\ &\le \sum_{\phi=1}^{t} \sum_{j\in\phi} \frac{1}{\mu_\phi(e_0)} \chi_j(e_0) C_j(\pi_j) + O\left(\epsilon H t + \frac{H \log \Delta}{\epsilon}\right). \end{aligned}$$

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*

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

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 prefix(v) (this was proved in claim 3.2) and the portion of the path following w is distributed according to suffix(w) (this follows from the definition of $\chi_j(e)$ and of suffix(w)). Moreover, for any edge e',

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

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

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

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

□

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

$$\sum_{\phi=1}^{t} \sum_{e \in \Delta(v)} \sum_{j \in \phi} \frac{p_\phi(e)}{\mu_\phi(e)} \cdot \frac{\mu_\phi(e)}{\tau} \cdot (A_j(v) + B_j(w) + \mathbf{E}_\phi[C_j(e)])$$

$$= \frac{1}{\tau} \sum_{j=1}^{T} \sum_{e \in \Delta(v)} p_\phi(e) \left(A_j(v) + B_j(w) + \mathbf{E}_\phi[C_j(e)]\right),$$

while the sum on the right side will become

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

$$= \frac{1}{\tau} \sum_{j=1}^{T} \left(A_j(v) + B_j(w_0) + \mathbf{E}_\phi[C_j(e_0)]\right).$$

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

$$\frac{1}{\tau} \sum_{j=1}^{T} \sum_{e \in \Delta(v)} p_\phi(e) \left(\mathbf{E}_\phi[C_j(e)] + B_j(w)\right)$$

$$\leq \frac{1}{\tau} \sum_{j=1}^{T} \left(\mathbf{E}_\phi[C_j(e_0)] + B_j(w_0)\right) + O\left(\epsilon H t + \frac{H \log \Delta}{\epsilon}\right).$$

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

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

Thus we have derived

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

3.6.2 Global performance guarantee

Claim 3.4. *Let Δ denote the maximum outdegree in G. For all v,*

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

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. *Setting $\delta = \epsilon = \left(\frac{\tilde{m} \log(\Delta) \log(\tilde{m}T)}{T}\right)^{1/3}$, the algorithm suffers regret*

$$O\left(H^{7/3} (m \log(\Delta) \log(mHT))^{1/3} T^{2/3}\right).$$

Proof. The exploration steps are a δ fraction of all time steps, and each contributes at most H to the regret, so they contribute at most $\delta T H$ 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

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

Plugging in the parameter settings specified in the theorem, we obtain the desired conclusion. □

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.

5. REFERENCES

[1] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. 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.

[2] Baruch Awerbuch and Yishay Mansour. Online learning of reliable network paths. In *PODC*, 2003.

[3] Avrim Blum, Geoff Gordon, and Brendan McMahan. Bandit version of the shortest paths problem. Perosnal communication, July 2003.

[4] Adam Kalai and Santosh Vempala. Efficient algorithms for the online decision problem. In *Proc. of 16th Conf. on Computational Learning Theory, Wash. DC*, 2003.

[5] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–260, 1994.

[6] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[7] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994. A preliminary version appeared in FOCS 1989.

[8] Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. In *COLT Proceedings*, 2002.