Online Oblivious Routing*

Nikhil Bansal Avrim Blum Shuchi Chawla Adam Meyerson

Carnegie Mellon University, Pittsburgh, PA, USA.

{nikhil, avrim, shuchi, adam}@cs.cmu.edu

ABSTRACT

We consider an online version of the oblivious routing problem. Oblivious routing is the problem of picking a routing between each pair of nodes (or a set of flows), without knowledge of the traffic or demand between each pair, with the goal of minimizing the maximum congestion on any edge in the graph. In the online version of the problem, we consider a "repeated game" setting, in which the algorithm is allowed to choose a new routing each night, but is still oblivious to the demands that will occur the next day. The cost of the algorithm at every time step is its competitive ratio, or the ratio of its congestion to the minimum possible congestion for the demands at that time step.

We present an algorithm that is $(1+\epsilon)$ competitive with respect to the best algorithm that uses a single routing for the entire sequence of days (known as the optimal static routing). Our result is a strengthening of the recent result of Azar et al [4], who gave a polynomial time algorithm to find an oblivious routing with the best possible competitive ratio, in that our algorithm achieves a competitive ratio arbitrarily to close to that of Azar et al [4], while at the same time performing nearly as well as the optimal static routing for the given sequence of demands[1].

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms**]: General

Keywords

Online Algorithms, Networks, Competitive Analysis, Oblivious Routing, Congestion, Gradient Descent

*This research was supported in part by NSF grants CCR-0105488, NSF-ITR CCR-0122581, NSF-ITR IIS-0121678, and an IBM Graduate Fellowship.

[1]Our work was done independently, but subsequent to that of Azar et al [4].

SPAA'03, June 7–9, 2003, San Diego, California, USA.

1. INTRODUCTION

Routing virtual circuits in undirected networks is a widely studied problem. Given a graph G, an instance of the problem is a sequence of requests, where each request consists of a source-sink pair (i, j) and a bandwidth requirement $d(i, j)$. For each request, the algorithm is required to specify a flow or path for routing traffic from i to j. The goal is to minimize edge congestion, or the total bandwidth consumed on any link. In this paper we will be considering only the fractional version of this problem (a routing is a collection of *flows*).

Algorithms designed for this problem can be broadly classified into two kinds, *oblivious* and *adaptive*. An oblivious algorithm is one which determines a collection of flows, one for each source-sink pair (i, j), without any knowledge of the demands. An adaptive algorithm on the other hand, on receiving the demand $d(i, j)$, picks a flow for (i, j) based on demands seen previously. As expected, adaptive algorithms can obtain a "better quality" routing in comparison to oblivious algorithms. However, oblivious algorithms are attractive in that they can be computed in advance, have low implementation overhead and can be implemented in a fully distributed manner.

We use competitive analysis to judge the performance of an algorithm. Given the sequence of demands d, let r_d^* be the routing for which the maximum edge congestion is minimized. An algorithm (or, routing r) is called α-competitive with respect to r_d^*, if the maximum congestion under r is at most α times that under r_d^*.

Räcke[16] showed that for every undirected graph G, there exists an oblivious routing that is $O(\log^3 n)$ competitive for *any* set of demands. Very recently, Azar et al. [4] gave a polynomial time algorithm that, given the graph G, computes the routing with the best possible competitive ratio for that graph using the ellipsoid method. We call this routing the *minimax optimal* routing $\rho(G)$, and its competitive ratio, the minimax optimal cost. We use this terminology because the problem can naturally be thought of as an (exponential-size) matrix game, in which the rows correspond to routings, the columns correspond to sets of demands, and the matrix entries contain the competitive ratio of that algorithm on that input.

In this paper we consider a strengthening of Azar et al's result in the following way. Suppose that each day we are faced with an oblivious routing problem on the same graph G. As before, we want to be competitive against the optimal routing, irrespective of the set of demands. However, if it turns out the demands do have some particular structure, or come from some non-adversarial distribution, then we would like our algorithm to get the best possible competitive ratio on *those types* of demands.

We formulate the problem as the following online game: We have a network where requests are to be routed obliviously. However, the routing can be changed from day to day. On the morning of day t, the network administrator chooses an "oblivious routing of the day", given by the routing r^t. During the day requests arrive according to the vector of demands d^t and are routed according to r^t. At the end of the day the administrator incurs a cost c^t, which is equal to the competitive ratio of the routing r^t with respect to the optimal routing for demand vector d^t. The goal of the administrator is to minimize the average competitive ratio over all days.[2]

It is clear that choosing the minimax optimal routing ρ on each day guarantees a cost (competitive ratio) which is no more than the minimax optimal cost. Moreover, this holds for every possible demand sequence $\{d^1, d^2, \ldots\}$. However, it could be the case that for a typical demand sequence $\mathcal{D} = \{d^1, d^2, \ldots\}$, using some routing $r^*_{\mathcal{D}}$ every day incurs a significantly lower total cost than using ρ. Thus, an administrator using the minimax optimal routing would wish he had used $r^*_{\mathcal{D}}$. We call $r^*_{\mathcal{D}}$ the static optimal routing for demand sequence $\mathcal{D}$. For example, if all the demands d^t are the same, then the optimal routing is simply the one which minimizes the congestion on d^t, or $r^*_{d^t}$.

We give an online algorithm that runs in polynomial time and produces a sequence of routings r^t such that for every sequence $\mathcal{D}$ of demands the total cost incurred by the algorithm is no more than $(1+\epsilon)$ times the cost of the static optimum routing $r^*_{\mathcal{D}}$, so long as the sequence is sufficiently (polynomially) long in n and $1/\epsilon$. Notice that since the static optimal cost never exceeds the cost of ρ on $\mathcal{D}$, this implies that the total cost of the routing produced by the algorithm is never more than $(1+\epsilon)$ times the total cost of ρ. However, since the cost of the static optimum routing can be significantly better than the cost of ρ for some set of demands $\mathcal{D}$, the cost of routing produced by our algorithm could be significantly better than ρ as well.

It is interesting to compare our technique with that of Azar et al. The latter use the Ellipsoid algorithm on the space of all feasible routings. At every step, the routing corresponding to the center of the ellipsoid is considered. The algorithm constructs, in polynomial time, the worst case demand for that routing at that step. (That is, the separation oracle for the Ellipsoid algorithm is a simulated optimal adversary for the matrix game). If the congestion under this routing does not exceed the minimax optimal $\rho(G)$, then the algorithm terminates. Otherwise, the worst case demand acts as a violating constraint and reduces the space of the routings geometrically. Our algorithm on the other hand, uses a variant of the gradient descent algorithm, that also receives a set of demands (not necessarily worst case) at every time step. Although the gradient descent algorithm also converges to the minimax solution if worst case demands are given at every time step, it does so at a much slower rate compared to Ellipsoid – $O(\frac{1}{\epsilon^2})$ steps for a $(1+\epsilon)$ approximation as opposed to the $O(\log\frac{1}{\epsilon})$ steps taken by Ellipsoid. However, the gradient descent algorithm has the additional property that it converges to the best solution for that sequence of demands, whereas Ellipsoid does not. Thus, by giving up a $(1+\epsilon)$ factor, we are able to achieve this additional guarantee, and it is in this respect that our algorithm outperforms the Ellipsoid algorithm.

Note that our goals follow a classic line of research in game theory and machine learning, namely that of minimizing regret in repeated games [5, 10, 14, 9, 2]. Freund and Schapire [9] point out that the Weighted-Majority (WM) algorithm of Littlestone and Warmuth [14] can be used for repeated play in any 2-player zero-sum game, and will perform nearly as well as the best fixed strategy in hindsight. However, the WM algorithm involves placing weights on the strategies of the player using it (which are then dynamically adjusted over time) and in our case there are exponentially many such strategies. So instead we will draw on recent work of Kalai and Vempala [12] and Zinkevich [21] who give algorithms that are able to achieve the same types of guarantees and yet, as we will show, can be applied in polynomial time for the routing problem.

1.1 Related Work

The routing problem has been widely studied in both adaptive and oblivious models. The best known adaptive algorithm for general graphs is due to Aspnes et al[1], who give a $\log n$ competitive centralized algorithm. Awerbuch et al[3] achieve the same competitive ratio in a distributed setting.

In the oblivious model, it is well known that deterministic approaches (which must select a single path for each demand) perform very poorly in the worst case [6, 11]. This motivates the use of randomization. A randomized oblivious routing algorithm specifies a probability distribution on the paths between i and j for every (i, j) pair. We will view this as selecting a multi-commodity flow on the graph, and routing each packet probabilistically according to its flow densities.

Until recently, oblivious routing algorithms with good worst-case guarantees were known only for specific types of networks [18, 13]. However, Räcke [16] recently obtained a rather surprising result which implies the existence of a good randomized oblivious routing for *every* undirected graph. Azar et al [4] extend this work to give an algorithm based on the ellipsoid method that finds this minimax optimal routing in polynomial time.

Our online oblivious variant of the problem is closely related to the online convex programming problem, defined and studied by Kalai and Vempala [12] and Zinkevich[21]. The online convex programming problem is defined as follows: At each time step the algorithm is required to pick a vector $x^t \in \mathcal{F}$ where $\mathcal{F}$ is a convex set. The adversary then presents the algorithm with a convex function c^t. The cost incurred by the algorithm is $c^t(x^t)$, and the objective of the algorithm is to minimize the sum of costs over all time steps. This setting is closely related to the problem of prediction and regression under linear loss functions (see [8, 19, 20] and references therein), and the previously-mentioned problem of designing nearly-optimal strategies for repeated games [17, 7]. We review the algorithm of Zinkevich [21] in detail in Section 3 and adapt it to our setting.

The framework of online convex programming is general enough that it allows us to consider several interesting ex-

[2] We also consider other objectives, such as the sum of squares of the competitive ratios, or minimizing the average subject to maintaining a worst-case guarantee of never exceeding twice the minimax optimal.

tensions of the problem. For example, instead of minimizing the total cost (in our case, the competitive ratio) over all days, one can consider minimizing some convex function of these costs, that penalizes high costs very heavily. Alternatively, we can consider minimizing the sum of costs subject to the constraint that the cost incurred by the algorithm is no more than twice the minimax optimal cost. We discuss these extensions in Section 5.

The rest of this paper is organized as follows: We begin with a description of the problem and notation in section 2. Section 3 describes an algorithm for the online convex programming problem due to Zinkevich[21]. In section 4, we present our main result: a $(1+\epsilon)$-competitive algorithm for the online oblivious routing problem. In section 5, we extend our basic algorithm to other cost functions and optimization objectives. We conclude in section 6.

2. FRAMEWORK AND NOTATION

Let $G=(V,E)$ be a directed graph on n vertices. For a vertex v, $IN(v)$ denotes the set of incoming edges to v, and $OUT(v)$ denotes the set of outgoing edges from v. Let V^2 denote the set of commodities $\{(i,j) : i,j \in V, i \neq j\}$.

A route for commodity $(i,j) \in V^2$ is a unit flow from vertex i to vertex j. In other words, it is a function $r : E \rightarrow [0,1]$ satisfying the following linear inequalities:

$$\begin{array}{rcll} \sum_{e \in OUT(v)} r(e) - \sum_{e \in IN(v)} r(e) & = & 0 & \forall v \neq i,j \\ \sum_{e \in OUT(i)} r(e) - \sum_{e \in IN(i)} r(e) & = & 1 & \\ \sum_{e \in OUT(j)} r(e) - \sum_{e \in IN(j)} r(e) & = & -1 & \end{array} \quad (1)$$

A **routing** is a collection of $n(n-1)$ unit routes, one for each commodity, $r = (r_{i,j})_{(i,j)\in V^2}$. We denote by $r(e)$ the vector of values of $r_{i,j}(e)$ for each $(i,j) \in V^2$. Let $\mathcal{F}$ denote the set of all valid routings in the graph G. Note that $\mathcal{F}$ is a convex set, since any convex combination of valid routings is also a valid routing.

A **demand vector** is a vector of $n(n-1)$ non-negative terms. We will index such a vector by pairs $(i,j) \in V^2$.

Given a demand vector d and a routing r, the **flow** on an edge $e \in E$ is given by $f_{d,r}(e) = \sum_{(i,j)\in V^2} d(i,j) r_{i,j}(e) = d \cdot r(e)$, where $(\cdot)$ denotes the dot product between two vectors. The **congestion** of a routing r, given a demand vector d, is the maximum flow on any edge e: $\mathcal{C}_d(r) = \max_{e\in E} f_{d,r}(e)$.

The **cost** incurred by a routing r on demand vector d is given by the ratio of the routing's congestion to the minimal possible congestion for demand vector d: $\textsc{cost}_d(r) = \frac{\mathcal{C}_d(r)}{\min_{r'} \mathcal{C}_d(r')}$. Note that the cost of any routing is at least 1.

The oblivious routing problem is to find a routing ρ that achieves the minimum possible cost over all demand vectors. That is, $\rho = \mathrm{argmin}_r \max_d \textsc{cost}_d(r)$. We call ρ the **minimax optimal routing**. This is a widely studied notion of optimality for the routing problem and recently Azar et al[4] developed a polynomial time algorithm to find this routing for general networks.

We consider an online version of the problem in this paper. The **online oblivious routing** problem is stated as follows. At each time step t, the algorithm fixes a routing r^t, that is to be used to route demand in that time step. The algorithm then receives a demand vector d^t. The cost of the algorithm is given by $\sum_t \textsc{cost}_{d^t}(r^t)$, where $\textsc{cost}_{d^t}(r^t)$ is the cost at time step t.

We use competitive analysis to study the performance of our algorithms for oblivious routing. Given a sequence of demands $\mathcal{D} = \{d^t\}$, let $r^*_{\mathcal{D}}$ be the routing that minimizes $\sum_t \textsc{cost}_{d^t}(r)$. We call such a routing the **static optimal routing for demand sequence $\mathcal{D}$.**

We call an algorithm α-competitive with respect to a static optimal routing if it satisfies the following:

$$\sum_t \textsc{cost}_{d^t}(r^t) \leq \alpha \sum_t \textsc{cost}_{d^t}(r^*_{\mathcal{D}}) + \beta \qquad \forall \mathcal{D}$$

where β is a constant independent of $\mathcal{D}$. The main result of this paper is as follows:

THEOREM 1. *After $\frac{n^6}{\epsilon^2}$ steps, the Greedy Projection algorithm is $(1+4\epsilon)$-competitive with respect to the optimal static routing.*

Note that by definition, $\sum_t \textsc{cost}_{d^t}(r^*_{\mathcal{D}}) \leq \sum_t \textsc{cost}_{d^t}(\rho)$. Thus any algorithm that is α-competitive with respect to a static optimal routing, is also α-competitive with respect to the minimax optimal routing ρ.

3. ONLINE CONVEX PROGRAMMING

In this section, we review some work on Online Convex Programming, that is useful in our algorithms and analysis. We begin with some definitions.

A set $F \in I\!R^n$ is convex if for all $x,y \in F$ and for all $\lambda \in [0,1]$, $\lambda x + (1-\lambda)y \in F$. For a convex set F, a function $f : F \rightarrow I\!R$ is convex if for all $x,y \in F$ and all $\lambda \in [0,1]$, $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$.

An online convex programming problem consists of a feasible set $F \in I\!R^n$ and a sequence of cost functions $g = \{g^0, g^1, \ldots, g^T\}$ where each $g^t : F \rightarrow I\!R$ is a convex function. At every step, the online algorithm is required to pick an element $x^t \in F$, and is then presented with the function g^t. The algorithm incurs a cost of $g^t(x^t)$ at time step t. The goal of the online algorithm is to minimize the cost $\sum_t g^t(x^t)$.

Given a sequence of cost functions g, let the static optimal solution $x^*_g \in F$ be the element minimizing $\sum_t g^t(x)$. The **regret** of an algorithm is the difference between costs of the algorithm and the static optimal solution for cost functions g.

$$\textsc{regret}(ALG) = \sum_t g^t(x^t) - \sum_t g^t(x^*_g)$$

Zinkevich[21] give a deterministic algorithm for the online convex programming problem, that achieves a low regret of $O(\sqrt{T})$, with the constant depending on F and g. Before describing the algorithm, we need some more notation. Let $\|x\|$ denote the length $\sqrt{x \cdot x}$ of a vector x. Let $\delta(x,y) = \|x-y\|$ denote the distance between two vectors $x,y \in F$, and $\|F\| = max_{x,y}\delta(x,y)$ be the diameter of set F. Let η_t denote the learning rate of the algorithm, to be specified later. We denote by $\nabla g(x)$ the gradient of a function g at x.

The Greedy Projection Algorithm. The algorithm starts with an arbitrary vector $x^0 \in F$. At each step $t+1$, the algorithm first performs a gradient descent – it moves in the direction of the previously seen cost function. In case this point lies outside the feasible region F, the algorithm projects it back into the set F, by picking a point in F

closest to this point. The following equations describe the algorithm:

$$\begin{aligned} y^t &= x^t - \eta_t \nabla g^t(x^t) \\ x^{t+1} &= \arg\min_{x \in F} \delta(x, y^t) \end{aligned}$$

We have the following theorem:

THEOREM 2. *For the online convex programming problem $(F, \{g^0, g^1, \dots\})$, the regret of the Greedy Projection algorithm with $\eta_t = t^{-1/2}$ after T time steps is at most*

$$\sqrt{T+1}\left(\frac{1}{2}\|F\|^2 + \max_{t, x \in F} \|\nabla g^t(x)\|^2\right)$$

Note that although the performance of the above algorithm is stated in terms of regret, we can convert this to a competitive ratio by using a lower bound on the cost of static optimal solution. Because the result [21] is not yet published, we provide a self-contained proof for the case of linear cost functions g (which is all we need for our main result) inside the proof of Lemma 8.

4. ONLINE OBLIVIOUS ROUTING

In this section we show that the online oblivious routing problem can be converted to an online convex programming problem.

Recall that the cost of the online algorithm at step t is given by $\frac{\mathcal{C}_{d^t}(r^t)}{\min_r \mathcal{C}_{d^t}(r)}$. Let $\kappa(d^t) = \min_r \mathcal{C}_{d^t}(r)$. Consider the demand vector $\overline{d}^t = \frac{d^t}{\kappa(d^t)}$. Notice that the optimal congestion on the demand vector $\overline{d}^t$ is exactly 1. Consequently, although the flow is reduced by a factor of $\kappa(d^t)$, the cost remains the same: $\text{COST}_{d^t}(r^t) = \text{COST}_{\overline{d}^t}(r^t)$. Henceforth we will assume that the algorithm receives the demand vector $\overline{d}^t$ at time step t.

The cost incurred by the algorithm at any step is a maximum over $|E|$ convex functions – each representing the flow on an edge $e \in E$. We next describe how to convert this function into a linear function. Let e^t be the maximum congested edge at time t, $e^t = \text{argmax}_{e \in E} f_{\overline{d}^t, r}(e)$. Define c^t to be an $n(n-1)|E|$ dimensional vector given by the following:

$$c^t_{e,i,j} = \begin{cases} \overline{d}^t(i,j), & \text{if } \; e^t = e \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

The following lemma characterizes the cost of the algorithm in terms of c^t.

LEMMA 3. *The cost of the algorithm at step t is given by $\text{COST}_{d^t}(r^t) = c^t \cdot r^t$.*

Proof: By definition of c^t, $c^t \cdot r^t = \overline{d}^t \cdot r^t(e^t) = f_{\overline{d}^t, r^t}(e^t)$. The latter is equal to $\max_{e \in E} f_{\overline{d}^t, r^t}(e) = \text{COST}_{d^t}(r^t)$ by the definition of e^t. ∎

Now consider the static optimal routing $r^*_{\mathcal{D}}$.[3] The cost incurred by r^* with respect to $c^1, c^2, \dots$ is

$$\sum_t c^t \cdot r^* = \sum_t \overline{d}^t \cdot r^*(e^t) \le \sum_t \max_{e \in E} f_{\overline{d}^t, r^*}(e) = \sum_t \text{COST}_{d^t}(r^*)$$

So we get the following lemma:

[3]We will drop the subscript $\mathcal{D}$ when it is obvious from the context.

LEMMA 4. *The cost of the static optimal routing r^* at step t is bounded below by $c^t \cdot r^*$.*

Lemmas 3 and 4 together give the following result.

THEOREM 5. *An algorithm that is α-competitive for the online convex programming problem $(\mathcal{F}, c)$ as defined above, is α-competitive against the static optimal routing for the demand vectors $\mathcal{D}$.*

Now our algorithm is an application of the Greedy Projection algorithm to the problem $(\mathcal{F}, c)$. Note that $\nabla(c^t \cdot x) = c^t$ $\forall t$. The algorithm is as follows:

1. Pick $r^0 \in \mathcal{F}$ arbitrarily
2. At step $t+1$:
   (a) Compute c^t from d^t using equation (2).
   (b) Let $y^t = r^t - \eta_t c^t$ with $\eta_t = \frac{2}{\sqrt{t}}$.
   (c) Pick $r^{t+1} \in \mathcal{F}$ to be a $(1+\epsilon_t)$ approximation to $\text{argmin}_{x \in \mathcal{F}} \|x - y^t\|^2$ using the Projection algorithm defined below, where $\epsilon_t = \frac{1}{t}$.

We complete our description of the algorithm by giving a projection algorithm for step 2c.

The projection algorithm. Given a vector y and real number ϵ, the algorithm finds a vector $x \in \mathcal{F}$ such that the squared distance of this vector from y is within $1+\epsilon$ of the minimum squared distance from y to $\mathcal{F}$. The vector x is given by the following quadratic program:

$$\min \; (x-y) \cdot (x-y) \quad \text{subject to} \quad x \in \mathcal{F}$$

This can be rewritten as the following Semi-Definite Program:

$$\min \; t \quad \text{subject to} \qquad x \in \mathcal{F}$$
$$\begin{bmatrix} I & (x-y) \\ (x-y)^T & t \end{bmatrix} \ge 0$$

The above program can be solved to within as small an error as desired in polynomial time, using the ellipsoid method for semi-definite programming [15]:

THEOREM 6 (THEOREM 3.7 IN [15]). *Let K be a convex body in $\mathbb{R}^n$ with $B(r) \subseteq K \subseteq B(R)$, for some $0 \le r \le R$, where $B(r)$ is a ball of radius r around the origin. Assume that we have a separation oracle for K. Let $c \in \mathbb{R}^n$ and $\epsilon > 0$ be given. Then, we can find $x \in K$ with $c^T x \ge c^T z - \epsilon$, $\forall z \in K$, using the ellipsoid algorithm, with the number of calls to the oracle and computation time bounded by a polynomial in $\log(R/r)$, $\log(1/\epsilon)$, and n.*

This completes the description of our algorithm. We start the analysis with a technical lemma for the projection procedure (proven in the appendix).

LEMMA 7. *Given a vector y and $\epsilon < \frac{1}{5}$, let r be the solution returned by the Projection algorithm. Then for any $x \in \mathcal{F}$, we have $\|x - r\|^2 \le (1+3\epsilon)\|x - y\|^2$.*

Our main result is as follows.

LEMMA 8. $\sum_t (r^t - r^*) \cdot c^t \leq 4n^3 \sqrt{T+1}$

Proof: We get that over T steps,

$$\begin{aligned}
& \sum_{t \leq T} (r^t - r^*) \cdot c^t \\
= & \sum_{t \leq T} \left(\frac{r^t + y^t}{2} - r^* \right) \cdot c^t + \sum_{t \leq T} \left(\frac{r^t - y^t}{2} \right) \cdot c^t \\
= & \frac{1}{2} \sum_{t \leq T} \frac{1}{\eta_t} (r^t + y^t - 2r^*) \cdot (r^t - y^t) + \frac{1}{2} \sum_{t \leq T} \eta_t \|c^t\|^2 \\
= & \frac{1}{2} \sum_{t \leq T} \frac{1}{\eta_t} \left(\|r^t - r^*\|^2 - \|y^t - r^*\|^2 \right) + \frac{1}{2} \sum_{t \leq T} \eta_t \|c^t\|^2 \\
\leq & \frac{1}{2} \sum_{t \leq T} \frac{1}{\eta_t} \left(\|r^t - r^*\|^2 - (1 - 3\epsilon) \|r^{t+1} - r^*\|^2 \right) \\
& + \frac{1}{2} \sum_{t \leq T} \eta_t \|c^t\|^2 \\
\leq & \frac{1}{2\eta_T} \|\mathcal{F}\|^2 + \frac{3}{2} \|\mathcal{F}\|^2 \sum_{t \leq T} \frac{1}{\eta_t} \epsilon_t + \frac{\max \|c^t\|^2}{2} \sum_{t \leq T} \eta_t
\end{aligned}$$

Here, the second step follows from the fact that $c^t = \frac{1}{\eta_t}(r^t - y^t)$, and the fourth inequality follows from lemma 7. The fifth step follows from the definition of $\|\mathcal{F}\|^2$.

Using $\eta_t = \frac{2}{\sqrt{t}}$ and $\epsilon_t = \frac{1}{t}$, we get that $\sum_t (r^t - r^*) \cdot c^t \leq \sqrt{T+1} \|\mathcal{F}\|^2 + 2 \max \|c^t\|^2 \sqrt{T+1}$.

Next we bound the terms $\max \|c^t\|^2$ and $\|\mathcal{F}\|^2$. First consider $\|\mathcal{F}\|^2$. This is less than $2 \max_{r \in \mathcal{F}} \|r\|^2$. Each term in the vector $r \in \mathcal{F}$ has value at most 1, by definition. Moreover, the sum of all terms, or the total flow in this routing r, is at most n^3. This is because each commodity with demand 1 contributes at most n to the total flow in the graph[4]. Now $\|r\|^2 \leq \max_{i,j,e} r_{i,j}(e) \sum_{i,j,e} r_{i,j}(e) \leq n^3$. So we have $\|\mathcal{F}\|^2 \leq 2n^3$.

Similarly, c^t has n^2 non-zero terms, corresponding to the scaled demands $\overline{d}^t$. Since the flow on any edge due to the scaled demands is at most 1, each scaled demand is at most n. Moreover, the total demand can be at most n^2. Thus we get $\max \|c^t\|^2 \leq n^3$.

Using $\|\mathcal{F}\|^2 \leq 2n^3$ and $\|c^t\|^2 \leq n^3$, we get $\sum_t (r^t - r^*) \cdot c^t \leq 4n^3 \sqrt{T+1}$. ■

Using the fact that the competitive ratio of the optimal routing is at least 1 in every step, we get that the algorithm converges in a polynomial number of steps:

Proof of Theorem 1: From theorem 5 and lemma 8 we have $\text{COST}(ALG) - \text{COST}(r^*) \leq 4n^3 \sqrt{T+1}$. Using the fact that the cost of r^* is at least 1 at every step, the algorithm converges after $\frac{1}{\epsilon^2} n^6$ steps, and we get $\text{COST}(ALG) \leq (1 + 4\epsilon)\text{COST}(r^*)$. ■

5. EXTENSIONS

Observe that although the algorithm described in the previous section achieves a good competitive ratio with respect to the static optimal solution in the long term, it may perform very poorly in the short term. Motivated by this observation, in this section we extend our result to other cost functions that penalize solutions which are expensive in the short term.

We first consider penalizing high congestion more aggressively by charging the algorithm the p-th power of the maximum congestion at any time step. In other words, the cost at time t is given by $\text{COST}_{d^t}(r^t) = \left(\frac{C_{d^t}(r^t)}{\kappa(d^t)} \right)^p$.

As before, we need to define an appropriate cost vector c^t. The high level idea is as follows. We pick a vector c^t that has the property that the cost of the algorithm is equal to $c^t \cdot r^t$, whereas the cost of any other routing r is at least $c^t \cdot r$. This would ensure, along the lines of lemmas 3 and 4, that an algorithm that is competitive with respect to $(\mathcal{F}, c)$, is also competitive with respect to the static optimal solution.

Now the cost of the algorithm can be written as $\text{COST}_{d^t}(r^t) = z\,(d \cdot r(e_r^*))^p$, where z is some constant. In other words, it is simply a polynomial of degree p in r^t. Thus, as before, an appropriate choice for c^t is the gradient of this polynomial at r^t. Now we can simply apply our Greedy Projection algorithm, as defined in the previous section, to the problem $(\mathcal{F}, c)$ and obtain a competitive solution as before. However, our rate of convergence is much slower in this case, because $\|c^t\|^2$ in this case can be as large as $O(n^{p+2})$. Picking the parameters η_t and ϵ_t optimally, we get that the Greedy Projection algorithm given a $(1+\epsilon)$ approximation to the static optimal routing in $O(\frac{1}{\epsilon^2} n^{5+p})$ steps.

Next we consider a more strict restriction on the algorithm: it should never incur a cost more than twice that of the minimax optimal cost. In other words, let $\text{COST}(\rho) = \max_d \text{COST}_d(\rho)$ be the worst possible cost of the minimax optimal strategy ρ. Then, the algorithm should never incur a cost more than $2\text{COST}(\rho)$. Note that in this case our routing is only guaranteed to be competitive with respect to the best static routing that does not incur a cost of more than $2\text{COST}(\rho)$ for any vector of demands.

In order to achieve this, the algorithm first computes the minimax optimal cost using the algorithm of Azar et al[4]. This is known to be at most $O(\log^3 n)$. Then, we run the Greedy Projection algorithm as before, with the modification of picking each routing from a smaller set $\mathcal{F}' \subseteq \mathcal{F}$. The set $\mathcal{F}'$ is given by the following system of inequalities in addition to the ones that define $\mathcal{F}$ (equation 1).

$$f_{d,x}(e) \leq 2\text{COST}(\rho) \quad \forall e \in E, d \in D \tag{3}$$

Note that the set $\mathcal{F}'$ is a set defined by (possibly infinitely many) linear inequalities. We can use the ellipsoid method, just as in Section 4, in the projection step of the algorithm. We get the same competitive ratio as given in theorem 1.

6. CONCLUSIONS

In this paper, we applied ideas from repeated games and online machine learning to the oblivious routing problem. We developed an algorithm that extends the result of Azar et al [4], in that it performs nearly as well as the best static routing, even if the demands turn out to be such that the best static routing is *better* than minimax optimal. It would be interesting to improve the rate of convergence of our algorithm, and to see how well it performs in practice.

[4]If routed over a single path, each commodity contributes at most n to that path, and a flow is simply a convex combination of paths.

7. REFERENCES

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[3] B. Awerbuch and Y. Azar. Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation. In *Foundations of Computer Science*, pages 240–249, 1994.

[4] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proc. of ACM Symposium on the Theory of Computation*, 2003, to appear.

[5] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6:1–8, 1956.

[6] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, 1985.

[7] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[8] N. Cesa-Bianchi. Analysis of two gradient-based algorithms for on-line regression. *Journal of Computer and System Sciences*, 59(3):392–411, 1999.

[9] Y. Freund and R. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[10] J. Hannan. Approximation to Bayes risk in repeated play. In M. Dresher, A. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games*, volume III, pages 97–139. Princeton University Press, 1957.

[11] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, 1990.

[12] A. Kalai and S. Vempala. Geometric algorithms for online optimization. Technical Report MIT-LCS-TR-861, MIT Laboratory for Computer Science, 2002.

[13] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.

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

[15] L. Lovász. Semidefinite programs and combinatorial optimization (lecture notes). http://research.microsoft.com/users/lovasz/semidef.ps.

[16] H. Räcke. Minimizing congestion in general networks. In *Foundations of Computer Science*, pages 43–53, 2002.

[17] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *16th Conference on Uncertainty in Artificial Intelligence*, 2000.

[18] L. G. Valiant and G. Brebner. Universal schemes for parallel communication. In *Proc. of ACM Symposium on the Theory of Computation*, pages 263–277, 1981.

[19] M. Warmuth and C. Gentile. Proving relative loss bounds for on-line learning algorithms using bregman divergences. In *Tutorial at Computational Learning Theory (COLT)*, 2000.

[20] M. Warmuth and J. Kivinen. Additive versus exponentiated gradient updates for linear prediction. *Journal of Information and Computation*, 132(1):1–64, 1997.

[21] M. Zinkevich. Online convex programming. *Manuscript, in submission*, 2002.

APPENDIX

Proof of Lemma 7: Consider a perpendicular from y to the line joining x and r. Let b be the point of intersection of the perpendicular and the line. Note that b lies between x and r, otherwise, $\|x-r\| < \|x-y\|$ and we are done.

Then, $b \in \mathcal{F}$ because $\mathcal{F}$ is a convex set. By the approximation guarantee of the projection algorithm, we have $\|y-r\| \leq \sqrt{1+\epsilon}\|y-b\|$. Thus, $\|b-r\| \leq \epsilon\|y-b\| \leq \epsilon\|y-x\|$, because $\|y-b\| \leq \|y-x\|$ by definition. Again by definition we have $\|x-b\| \leq \|y-x\|$. Thus, $\|x-r\| = \|x-b\| + \|b-r\| \leq (1+\epsilon)\|y-x\|$ and we get the result. ■