Online Convex Programming and Generalized Infinitesimal Gradient Ascent

Martin Zinkevich MAZ@CS.CMU.EDU
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

Convex programming involves a convex set $F \subseteq \mathbb{R}^n$ and a convex cost function $c : F \to \mathbb{R}$. The goal of convex programming is to find a point in F which minimizes c. In online convex programming, the convex set is known in advance, but in each step of some repeated optimization problem, one must select a point in F before seeing the cost function for that step. This can be used to model factory production, farm production, and many other industrial optimization problems where one is unaware of the value of the items produced until they have already been constructed. We introduce an algorithm for this domain. We also apply this algorithm to repeated games, and show that it is really a generalization of infinitesimal gradient ascent, and the results here imply that generalized infinitesimal gradient ascent (GIGA) is universally consistent.

1. Introduction

Convex programming is a generalization of linear programming, with many applications to machine learning. For example, one wants to find a hypothesis in a hypothesis space H that minimizes absolute error (Boyd & Vandenberghe, 2003), minimizes squared error (Hastie et al., 2001), or maximizes the margin (Boser et al., 1992) for the training set X. If the hypothesis space consists of linear functions, then these problems can be solved using linear programming, least-squares regression, and support vector machines respectively. These are all examples of convex programming problems.

Convex programming has other applications, such as nonlinear facility location problems (Boyd & Vandenberghe, 2003, pages 421-422), network routing problems (Bansal et al., 2003), and consumer optimization problems (Boot, 2003, pages 4–6). Other examples of linear programming problems are meeting nutritional requirements, balancing production and consumption in the national economy, and production planning(Cameron, 1985, pages 36–39).

Convex programming consists of a convex feasible set $F \subseteq \mathbb{R}^n$ and a convex (valley-shaped) cost function $c : F \to \mathbb{R}$. In this paper, we discuss online convex programming, in which an algorithm faces a sequence of convex programming problems, each with the same feasible set but different cost functions. Each time the algorithm must choose a point before it observes the cost function. This models a number of optimization problems including industrial production and network routing, in which decisions must be made before true costs or values are known[1]. This is a generalization of both work in minimizing error online (Cesa-Bianchi et al., 1994; Kivinen & Warmuth, 1997; Gordon, 1999; Herbster & Warmuth, 2001; Kivinen & Warmuth, 2001) and of the experts problem (Freund & Schapire, 1999; Littlestone & Warmuth, 1989).

In the experts problem, one has n experts, each of which has a plan at each step with some cost. At each round, one selects a probability distribution over experts. If $x \in \mathbb{R}^n$ is defined such that x_i is the probability that one selects expert i, then the set of all probability distributions is a convex set. Also, the cost function on this set is linear, and therefore convex. Repeated games are closely related to the experts problem.

In minimizing error online, one sees an unlabeled instance, assigns it a label, and then receives some error based on how divergent the label given was from the true label. The divergences used in previous work are

[1] We expand on the network routing domain at the end of this section. In particular, our results have been applied (Bansal et al., 2003) to solve the "online oblivous routing problem".

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

fixed Bregman divergences, e.g. squared error.

In this paper, we make no distributional assumptions[2] about the convex cost functions. Also, we make no assumptions about any relationships between successive cost functions. Thus, expecting to choose the optimal point at each time step is unrealistic. Instead, as in the analysis of the experts problem, we compare our cost to the cost of some other "offline" algorithm that selects a fixed vector. However, this other algorithm knows in advance all of the cost functions before it selects this single fixed vector. We formalize this in Section 2.1.

We present an algorithm for general convex functions based on gradient descent, called greedy projection. The algorithm applies gradient descent in $\mathbb{R}^n$, and then moves back to the set of feasible points. There are three advantages to this algorithm. The first is that gradient descent is a simple, natural algorithm that is widely used, and studying its behavior is of intrinsic value. Secondly, this algorithm is more general than the experts setting, in that it can handle an arbitrary sequence of convex functions, which has yet to be solved. Finally, in online linear programs this algorithm can in some circumstances perform better than an experts algorithm. While the bounds on the performance of most experts algorithms depends on the number of experts, these bounds are based on other criteria which may sometimes be lower. This relationship is discussed further in Section 4, and further comments on related work can be found in Section 5. The main theorem is stated and proven in Section 2.1.

Another measure of the performance of gradient projection is found in Section 2.2, where we establish results unlike those usually found in online algorithms. We establish that the algorithm can perform well, even in comparison to an agent that knows the sequence in advance and can move for some short distance. This result establishes that greedy projection can handle environments that are slowly changing over time and require frequent but small modifications to handle well.

The algorithm that motivated this study was infinitesimal gradient ascent (Singh et al., 2000), which is an algorithm for repeated games. First, this result shows that infinitesimal gradient ascent is universally consistent (Fudenberg & Levine, 1995), and secondly it shows that GIGA, a nontrivial extension developed here of infinitesimal gradient ascent to games with more than two actions, is universally consistent. GIGA is defined in Section 3.2, and the proof is similar to that in (Freund & Schapire, 1999).

Bansal et al. (2003) formulate an online oblivious routing problem as an online convex programming problem, and apply greedy projection to obtain good performance. In online oblivious routing, one is in charge of minimizing network congestion by programming a variety of routers. At the beginning of each day, one chooses a flow for each source-destination pair. The set of all such flows is convex. Then, an adversary chooses a demand (number of packets) for each source-destination pair. The cost is the maximum congestion along any edge that the algorithm has divided by the maximum congestion of the optimal routing given the demand.

The contribution of this paper is a general solution for a wide variety of problems, some solved, some unsolved. Sometimes, these results show new properties of existing algorithms, like IGA, and sometimes, this work has resulted in new algorithms, like GIGA. Finally, the flexibility to choose arbitrary convex functions has already resulted in a solution to a practical online problem (Bansal et al., 2003).

2. Online Convex Programming

Definition 1 *A set of vectors $S \subseteq \mathbb{R}^n$ is **convex** if for all $x, x' \in S$, and all $\lambda \in [0,1]$, $\lambda x + (1-\lambda)x' \in S$.*

Definition 2 *For a convex set F, a function $f : F \to \mathbb{R}$ is **convex** if for all $x, y \in F$, for all $\lambda \in [0,1]$,*

$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y)$$

If one were to imagine a convex function $\mathbb{R}^2 \to \mathbb{R}$, where the function described the altitude, then the function would look like a valley.

Definition 3 *A **convex programming problem** consists of a convex feasible set F and a convex cost function $c : F \to \mathbb{R}$. The **optimal solution** is the solution that minimizes the cost.*

Definition 4 *An **online convex programming problem** consists of a feasible set $F \subseteq \mathbb{R}^n$ and an infinite sequence $\{c^1, c^2, \dots\}$ where each $c^t : F \to \mathbb{R}$ is a convex function.*

*At each time step t, an **online convex programming algorithm** selects a vector $x^t \in F$. After the vector is selected, it receives the cost function c^t.*

Because all information is not available before decisions are made, online algorithms do not reach "solutions", but instead achieve certain goals. See Section 2.1.

[2]The assumptions we do make are listed in the beginning of Section 2.

Define $\| x \| = \sqrt{x \cdot x}$ and $d(x, y) = \| x - y \|$. Throughout the remainder of the paper we will make seven assumptions:

1. The feasible set F is **bounded**. There exists $N \in \mathbb{R}$ such that for all $x, y \in F$, $d(x, y) \leq N$.
2. The feasible set F is **closed**. For all sequences $\{x^1, x^2, \dots\}$ where $x^t \in F$ for all t, if there exists a $x \in \mathbb{R}^n$ such that $x = \lim_{t \to \infty} x^t$, then $x \in F$.
3. The feasible set F is **nonempty**. There exists an $x \in F$.
4. For all t, c^t is differentiable[3].
5. There exists an $N \in \mathbb{R}$ such that for all t, for all $x \in F$, $\| \nabla c^t(x) \| \leq N$.
6. For all t, there exists an algorithm, given x, which produces $\nabla c^t(x)$.
7. For all $y \in \mathbb{R}^n$, there exists an algorithm which can produce $\operatorname{argmin}_{x \in F} d(x, y)$. We define the projection $P(y) = \operatorname{argmin}_{x \in F} d(x, y)$.

Given this machinery, we can describe our algorithm.

Algorithm 1 ***Greedy Projection*** *Select an arbitrary $x^1 \in F$ and a sequence of learning rates $\eta_1, \eta_2, \dots \in \mathbb{R}^+$. In time step t, after receiving a cost function, select the next vector x^{t+1} according to:*

$$x^{t+1} = P\left(x^t - \eta_t \nabla c^t(x^t)\right).$$

The basic principle at work in this algorithm is quite clear if we consider the case where the sequence $\{c^1, c^2, \dots\}$ is constant. In this case, our algorithm is operating in an unchanging valley. The boundary of the feasible set is the edge of the valley. By proceeding along the direction opposite the gradient, we walk down into the valley. By projecting back into the convex set, we skirt the edges of the valley.

2.1. Analyzing the Performance of the Algorithm

What we would like to do is to prove greedy projection "works" for any sequence of convex functions, even if these convex function are unrelated to one another. In this scenario, we cannot hope choose a point x^i that minimizes c^i, because c^i can be anything. Instead we try to minimize regret.

[3]Although we make the assumption that c^t is differentiable, the algorithm can also work if there exists an algorithm that, given x, can produce a vector g such that for all y, $g \cdot (y - x) \leq c^t(y) - c^t(x)$.

We calculate our regret by comparing ourselves to an "offline" algorithm that has all of the information before it has to make any decisions, but is more restricted than we are in the choices it can make. For example, imagine that the offline algorithm has access to the first T cost functions at the beginning. However, it can only choose one vector $x \in F$. Then the offline algorithm can attempt to minimize the cost function $c(x) = \sum_{i=1}^{T} c^i(x)$. This is an offline convex programming problem. Regret is the difference between our cost and the cost of the offline algorithm. Average regret is the regret divided by T, the number of rounds.

Definition 5 *Given an algorithm A, and a convex programming problem $(F, \{c^1, c^2, \dots\})$, if $\{x^1, x^2 \dots\}$ are the vectors selected by A, then the* ***cost*** *of A until time T is*

$$C_A(T) = \sum_{t=1}^{T} c^t(x^t).$$

The cost of a static feasible solution $x \in F$ until time T is

$$C_x(T) = \sum_{t=1}^{T} c^t(x).$$

The regret of algorithm A until time T is

$$R_A(T) = C_A(T) - \min_{x \in F} C_x(T).$$

As the sequences get longer, the offline algorithm finds itself at less of an advantage. If the sequence of cost functions is relatively stationary, then an online algorithm can learn what the cost functions will look like in the future. If the sequence of cost functions varies drastically, then the offline algorithm will not be able to take advantage of this because it selects a fixed point. **Our goal is to prove that the average regret of Greedy Projection approaches zero.** In order to state our results about bounding the regret of this algorithm, we need to specify some parameters. First, let us define:

$$\begin{aligned} \| F \| &= \max_{x, y \in F} d(x, y) \\ \| \nabla c \| &= \max_{x \in F, t \in \{1, 2, \dots\}} \| \nabla c^t(x) \|. \end{aligned}$$

Here is the first result derived in this paper:

Theorem 1 *If $\eta_t = t^{-1/2}$, the regret of the Greedy Projection algorithm is:*

$$R_G(T) \leq \frac{\| F \|^2 \sqrt{T}}{2} + \left(\sqrt{T} - \frac{1}{2}\right) \| \nabla c \|^2$$

Therefore, $\limsup_{T \to \infty} R_G(T)/T \leq 0$.

The first part of the bound is because we might begin on the wrong side of F. The second part is a result of the fact that we always respond after we see the cost function.

Proof: First we show that without loss of generality, for all t there exists a $g^t \in \mathbb{R}^n$ such that for all x, $c^t(x) = g^t \cdot x$.

First, begin with arbitrary $\{c^1, c^2, \dots\}$, run the algorithm and compute $\{x^1, x^2, \dots\}$. Then define $g^t = \nabla c^t(x^t)$. If we were to change c^t such that for all x, $c^t(x) = g^t \cdot x$, the behavior of the algorithm would be the same. Would the regret be the same?

Because c^t is convex, for all x:

$$c^t(x) \geq (\nabla c^t(x^t)) \cdot (x - x^t) + c^t(x^t).$$

Set x^* to be a statically optimal vector. Then, because $x^* \in F$: $c^t(x^*) \geq g^t \cdot (x^* - x^t) + c^t(x^t)$. Thus:

$$\begin{aligned} c^t(x^t) - c^t(x^*) &\leq c^t(x^t) - \left(g^t \cdot (x^* - x^t) + c^t(x^t)\right) \\ &\leq g^t \cdot x^t - g^t \cdot x^* \end{aligned}$$

Thus the regret would be at least as much with the modified sequence of functions.

We define for all t, $y^{t+1} = x^t - \eta_t g^t$. Observe that $x^{t+1} = P(y^{t+1})$. We will attempt to bound the regret of not playing action x^* on round t.

$$\begin{aligned} y^{t+1} - x^* &= (x^t - x^*) - \eta_t g^t \\ (y^{t+1} - x^*)^2 &= (x^t - x^*)^2 - 2\eta_t (x^t - x^*) \cdot g^t \\ &\quad + \eta_t^2 \| g^t \|^2 \end{aligned}$$

Observe that in the expression $a^2 - 2ab + b^2$, a^2 is a potential, $2ab$ is the immediate cost, and b^2 is the error (within a factor of $2\eta_t$). We will now begin to fully flush out these properties. For all $y \in \mathbb{R}^n$, for all $x \in F$, $(y - x)^2 \geq (P(y) - x)^2$. Also, $\| g^t \| \leq \| \nabla c \|$. So

$$\begin{aligned} (x^{t+1} - x^*)^2 &\leq (x^t - x^*)^2 - 2\eta_t (x^t - x^*) \cdot g^t \\ &\quad + \eta_t^2 \| \nabla c \|^2 \\ (x^t - x^*) \cdot g^t &\leq \frac{1}{2\eta_t} \left((x^t - x^*)^2 - (x^{t+1} - x^*)^2\right) \\ &\quad + \frac{\eta_t}{2} \| \nabla c \|^2 \end{aligned}$$

Now, by summing we get:

$$\begin{aligned} R_G(T) &= \sum_{t=1}^{T} (x^t - x^*) \cdot g^t \\ &\leq \sum_{t=1}^{T} \frac{1}{2\eta_t} \left((x^t - x^*)^2 - (x^{t+1} - x^*)^2\right) \\ &\quad + \frac{\eta_t}{2} \| \nabla c \|^2 \\ &\leq \frac{1}{2\eta_1} (x^1 - x^*)^2 - \frac{1}{2\eta_T} (x^{T+1} - x^*)^2 \\ &\quad + \frac{1}{2} \sum_{t=2}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) (x^t - x^*)^2 \\ &\quad + \frac{\| \nabla c \|^2}{2} \sum_{t=1}^{T} \eta_t \\ &\leq \| F \|^2 \left(\frac{1}{2\eta_1} + \frac{1}{2} \sum_{t=2}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right)\right) \\ &\quad + \frac{\| \nabla c \|^2}{2} \sum_{t=1}^{T} \eta_t \\ &\leq \| F \|^2 \frac{1}{2\eta_T} + \frac{\| \nabla c \|^2}{2} \sum_{t=1}^{T} \eta_t \end{aligned}$$

Now, if we define $\eta_t = \frac{1}{\sqrt{t}}$, then

$$\begin{aligned} \sum_{t=1}^{T} \eta_t &= \sum_{t=1}^{T} \frac{1}{\sqrt{t}} \\ &\leq 1 + \int_{t=1}^{T} \frac{dt}{\sqrt{t}} \\ &\leq 1 + \left[2\sqrt{t}\right]_1^T \\ &\leq 2\sqrt{T} - 1 \end{aligned}$$

Plugging this into the above equation yields the result. ∎

2.2. Regret Against a Dynamic Strategy

Another possibility for the offline algorithm is to allow a small amount of change. For example, imagine that the path that the offline algorithm follows is of limited size.

Definition 6 *The* ***path length*** *of a sequence* $x^1, \dots, x^T$ *is:*

$$\sum_{i=1}^{T-1} d(x^i, x^{i+1}).$$

Define $\mathbb{A}(T, L)$ to be the set of sequences with T vectors and a path length less than or equal to L.

Definition 7 *Given an algorithm A and a maximum path length L, the **dynamic regret** $R_A(T, L)$ is:*

$$R_A(T, L) = C_A(T) - \min_{A' \in \mathbb{A}(T,L)} C_{A'}(T).$$

Theorem 2 *If η is fixed, the dynamic regret of the Greedy Projection algorithm is:*

$$R_G(T, L) \quad \leq \quad \frac{7\| F \|^2}{4\eta} + \frac{L\| F \|}{\eta} + \frac{T\eta\| \nabla c \|^2}{2}$$

The proof is similar to the proof of Theorem 1, and is included in the full version of the paper (Zinkevich, 2003).

3. Generalized Infinitesimal Gradient Ascent

In this section, we establish that repeated games are online linear programming problems, and an application of our algorithm is universally consistent.

3.1. Repeated Games

From the perspective of one player, a repeated game is two sets of actions A and B, and a utility function $u : A \times B \to \mathbb{R}$. A pair in $A \times B$ is called a **joint action**. For the example in this section, we will think of a matching game. $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$, where $u(a_1, b_1) = u(a_2, b_2) = u(a_3, b_3) = 1$, and everywhere else u is zero.

As a game is being played, at each step the player will be selecting an action at random based on past joint actions, and the environment will be selecting an action at random based on past joint actions. We will formalize this later.

A **history** is a sequence of joint actions. $H^t = (A \times B)^t$ is the set of all histories of length t. Define $H = \bigcup_{t=0}^{\infty} H^t$ to be the set of all histories, and for any history $h \in H$, define $|h|$ to be the length of that history. An example of a history is:

$$h = \{(a_3, b_1), (a_1, b_2), (a_2, b_3), (a_2, b_2), (a_2, b_2)\}$$

In order to access the history, we define h_i to be the ith joint action. Thus, $h_3 = (a_2, b_3)$, $h_{1,1} = a_3$ and $h_{5,2} = b_2$. The **utility** of a history $h \in H$ is:

$$u_{total}(h) = \sum_{i=1}^{|h|} u(h_{i,1}, h_{i,2}).$$

The utility of the above example is $u_{total}(h) = 2$. We can define what the history would look like if we replaced the action of the player with a_2 at each time step.

$$h^{*\to a_2} = \{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_2, b_2), (a_2, b_2)\}$$

Now, $u_{total}(h^{*\to a_2}) = 3$. Thus we would have done better playing this action all the time. The definition of **regret of not playing action** a for all $h \in H$, for all $a \in A$ is:

$$R^{*\to a}(h) = u_{total}(h^{*\to a}) - u_{total}(h)$$

In this example, the regret of not playing action a_2 is $R^{*\to a_2}(h) = 3 - 2 = 1$. This regret of not playing an action need not be positive. For example, $R^{*\to a_1}(h) = 1 - 2 = -1$. Now, we define the maximum regret, or just **regret**, to be:

$$R(h) = \max_{a \in A} R^{*\to a}(h).$$

Here $R(h) = 1$. The most important aspect of this definition of regret is that regret is a function of the resulting history, independent of the strategies that generated that history.

Now, we introduce the definition of the behavior and the environment. For any set S, define $\Delta(S)$ to be the set of all probabilities over S. For a distribution D and a boolean predicate P, we use the notation $\Pr_{x \in D}[P(x)]$ to indicate the probability that $P(x)$ is true given that x was selected from D.

A **behavior** $\sigma : H \to \Delta(A)$ is a function from histories of past actions to distributions over the next action of the player. An **environment** $\rho : H \to \Delta(B)$ is a function from the history of past actions to distributions over the next action of the environment. Define $H^\infty = (A \times B)^\infty$ to be the set of all infinite histories. We define $F_{\sigma,\rho} \in \Delta(H^\infty)$ to be the distribution over infinite histories where the player chooses its next action according to σ and the environment chooses its next action according to ρ. For all $h \in H^\infty$, define $h(t)$ to be the first t joint actions of h.

Definition 8 *A behavior σ is universally consistent if for any $\epsilon > 0$ there exists a T such that for all ρ:*

$$\Pr_{h \in F_{\sigma,\rho}} \left[\forall t > T, \frac{R(h(t))}{t} > \epsilon \right] < \epsilon.$$

After some time, with high probability the average regret is low. Observe that this convergence over time is uniform over all environments.

3.2. Formulating a Repeated Game as an Online Linear Program

For simplicity, suppose that we consider the case where $A = \{1, \ldots, n\}$. Before each time step in a repeated game, we select a distribution over actions. This can be represented as a vector in an n-standard closed simplex, the set of all points $x \in \mathbb{R}^n$ such that for all i, $x_i \geq 0$, and $\sum_{i=1}^{n} x_i = 1$. Define this to be F.

Since we have a utility u instead of cost c, we will perform gradient ascent instead of descent. The utility u is a linear function when the environment's action becomes known.

Algorithm 2 *(**Generalized Infinitesimal Gradient Ascent**) Choose a sequence of learning rates $\{\eta_1, \eta_2, \ldots\}$. Begin with an arbitrary vector $x^1 \in F$. Then for each round t:*

1. *Play according to x^t: play action i with probability x_i^t.*
2. *Observe the action $h_{t,2}$ of the other player and calculate:*

$$\begin{aligned} y_i^{t+1} &= x_i^t + \eta_t u(i, h_{t,2}) \\ x^{t+1} &= P(y^{t+1}) \end{aligned}$$

where $P(y) = \text{argmin}_{x \in F} d(x, y)$, as before.

Theorem 3 *Setting $\eta_t = \frac{1}{\sqrt{t}}$, GIGA is universally consistent.*

The proof is in the full version of the paper (Zinkevich, 2003), and we provide a sketch here. As a direct result of Theorem 1, we can prove that if the environment plays any fixed sequence of actions, our regret goes to zero. Using a technique similar to Section 3.1 of (Freund & Schapire, 1999), we can move from this result to convergence with respect to an arbitrary, adaptive environment.

4. Converting Old Algorithms

In this section, in order to compare our work with that of others, we show how one can naïvely translate algorithms for mixing experts into algorithms for online linear programs, and online linear programming algorithms into algorithms for online convex programs. This section is a discussion and no formal proofs are given.

4.1. Formal Definitions

We begin with defining the expert's problem.

Definition 9 *An **experts problem** is a set of experts $E = \{e_1, \ldots, e_n\}$ and a sequence of cost vectors $c^1, c^2, \ldots$ where for all i, $c^i \in \mathbb{R}^n$.*

*On each round t, an **expert algorithm (EA)** first selects a distribution $D^t \in \Delta(E)$, and then observes a cost vector c^t.*

We assume that the EA can handle both positive and negative values. If not, it can be easily extended by shifting the values into the positive range.

Definition 10 *An **online linear programming problem** is a closed convex polytope $F \subseteq \mathbb{R}^n$ and a sequence of cost vectors $c^1, c^2, \ldots$ where for all i,$c^i \in \mathbb{R}^n$.*

*On each round t, an **online linear programming algorithm (OLPA)** first plays a distribution $D^t \in \Delta(F)$, and then observes a cost vector c^t.*

An OLPA can be constructed from an EA, as described below.

Algorithm 3 *Define $v^1, \ldots, v^k$ to be the vertices of the polytope for an online linear program. Choose $E = \{e_1, \ldots, e_k\}$ to be the experts, one for each vertex.*

On each round t, receive a distribution D^t from the EA, and select vector v^i if expert e_i is selected. Define $c' \in \mathbb{R}^k$ such that $c'_i = c^t \cdot v^i$. Send EA the cost vector $c' \in \mathbb{R}^k$.

The optimal static vector must be a vertex of the polytope, because a linear program always has a solution at a vertex of the polytope. If the original EA can do almost as well as the best expert, this OLPA can do at least as well as the best static vector.

The second observation is that most EA have bounds that depend on the number of experts. The number of vertices of the convex polytope is totally unrelated to the diameter, so any normal expert's bound is incomparable to our bound on Greedy Projection.

There are some EA that begin with a distribution or uneven weighting over the experts. These EA may perform better in this scenario, because that one might be able to tweak the distribution such that it is spread evenly over the space (in some way) and not the experts, giving more weight to lonely vertices and less weight to clustered vertices.

4.2. Converting an OLPA to an Online Convex Programming Algorithm

There are two reasons that the algorithm described above will not work for an online convex program. The first is that an online convex program can have an

arbitrary convex shape as a feasible region, such as a circle, which cannot be described as the convex hull of any finite number of points.

The second reason is that a convex function may not have a minimum on the edge of the feasible set. For example, if $F = \{x : x \cdot x \leq 1\}$ and $c(x) = x \cdot x$, the minimum is in the center of the feasible set.

Now, this first issue is difficult to handle directly[4], so we will simply assume that the OLPA can handle the feasible region of the online convex programming problem. This can be either because that the OLPA can handle an arbitrary convex region as in Kalai and Vempala (2002), or because that the convex region of the convex programming problem is a convex polytope.

We handle the second issue by converting the cost function to a linear one. In Theorem 1, we find that the worst case is when the cost function is linear. This assumption depends on two properties of the algorithm; the algorithm is deterministic, and the only property of the cost function c^t that is observed is $\nabla c^t(x^t)$. Now, we form an Online Convex Programming algorithm in the following way.

Algorithm 4 *On each round t, receive D^t from the OLPA, and play $x^t = E_{X \in D^t}[X]$. Send the OLPA the cost vector $\nabla c^t(x^t)$.*

The algorithm is discrete and only observes the gradient at the point x^t, thus we can assume that the cost function is linear. If the cost function is linear, then:

$$E_{X \in D^t}[c^t(X)] = c^t(E_{X \in D^t}[X]).$$

x^t may be difficult to compute, and we address this issue in the full version of the paper.

5. Related Work

Kalai and Vempala (2002) have developed algorithms to solve online linear programming, which is a specific type of online convex programming. They are attempting to make the algorithm behave in a lazy fashion, changing its vector slowly, whereas here we are attempting to be more dynamic, as is highlighted in Section 2.2.

These algorithms were motivated by the algorithm of (Singh et al., 2000) which applies gradient ascent to repeated games. We extend their algorithm to games with an arbitrary number of actions, and prove universal consistency. There has been extensive work on regret in repeated games and in the experts domain, such as (Blackwell, 1956; Foster & Vohra, 1999; Foster, 1999; Freund & Schapire, 1999; Fudenberg & Levine, 1995; Fudenberg & Levine, 1997; Hannan, 1957; Hart & Mas-Colell, 2000; Hart & Mas-Colell, 2001; Littlestone & Warmuth, 1989). What makes this work noteworthy in a very old field is that it proves that a widely-used technique in artificial intelligence, gradient ascent, has a property that is of interest to those in game theory. As stated in Section 4, experts algorithms can be used to solve online online linear programs and online convex programming problems, but the bounds may become significantly worse.

There are several studies of online gradient descent and related update functions, for example (Cesa-Bianchi et al., 1994; Kivinen & Warmuth, 1997; Gordon, 1999; Herbster & Warmuth, 2001; Kivinen & Warmuth, 2001). These studies focus on prediction problems where the loss functions are convex Bregman divergences. In this paper, we are considering arbitrary convex functions, in problems that may or may not involve prediction.

Finally, in the offline case, (Della Pietra et al., 1999) have done work on proving that gradient descent and projection for arbitrary Bregman distances converges to the optimal solution.

6. Future Work

Here we deal with a Euclidean geometry: what if one considered gradient descent on a noneuclidean geometry, like (Amari, 1998; Mahony & Williamson, 2001)? It is also possible to conceive of cost functions that do not just depend on the most recent vector, but on every previous vector.

7. Conclusions

In this paper, we have defined an online convex programming problem. We have established that gradient descent is a very effective algorithm on this problem, because that the average regret will approach zero. This work was motivated by trying to better understand the infinitesimal gradient ascent algorithm, and the techniques developed we applied to that problem to establish an extension to infinitesimal gradient ascent that is universally consistent.

Acknowledgements

This work was supported in part by NSF grants CCR-0105488, NSF-ITR CCR-0122581, and NSF-ITR IIS-0121678. Any errors or omissions in the work are the

[4] One can approximate a convex region by a series of increasingly complex convex polytopes, but this solution is very undesirable.

sole responsibility of the author. We would like to thank Pat Riley for great help in developing the algorithm for the case of repeated games, Adam Kalai for improving the proof and bounds of the main theorem, and Michael Bowling, Avrim Blum, Nikhil Bansal, Geoff Gordon, and Manfred Warmuth for their help and suggestions with this research.

References

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*, 251–276.

Bansal, N., Blum, A., Chawla, S., & Meyerson, A. (2003). Online oblivious routing. *Fifteenth ACM Symposium on Parallelism in Algorithms and Architecture*.

Blackwell, D. (1956). An analog of the minimax theorem for vector payoffs. *South Pacific J. of Mathematics*, 1–8.

Boot, J. (2003). *Quadratic programming: Algorithms, anomolies, applications*. Rand McNally & Co.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Conference on Computational Learning Theory*.

Boyd, S., & Vandenberghe, L. (2003). Convex optimization. In press, available at `http://www.stanford.edu/~boyd/cvxbook.html`.

Cameron, N. (1985). *Introduction to linear and convex programming*. Cambridge University Press.

Cesa-Bianchi, N., Long, P., & Warmuth, M. K. (1994). Worst-case quadratic bounds for on-line prediction of linear functions by gradient descent. *IEEE Transactions on Neural Networks*, *7*, 604–619.

Della Pietra, S., Della Pietra, V., & Lafferty, J. (1999). *Duality and auxilary functions for Bregman distances* (Technical Report CMU-CS-01-109). Carnegie Mellon University.

Foster, D. (1999). A proof of calibration via Blackwell's approachability theorem. *Games and Economic Behavior* (pp. 73–79).

Foster, D., & Vohra, R. (1999). Regret in the on-line decision problem. *Games and Economic Behavior*, *29*, 7–35.

Freund, Y., & Schapire, R. (1999). Adaptive game playing using multiplicative weights. *Games and Economic Behavior* (pp. 79–103).

Fudenberg, D., & Levine, D. (1995). Universal consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, *19*, 1065–1089.

Fudenberg, D., & Levine, D. (1997). Conditional universal consistency. Available at http://ideas.repec.org/s/cla/levarc.html.

Gordon, G. (1999). *Approximate solutions to markov decision processes*. Doctoral dissertation, Carnegie Mellon University.

Hannan, J. (1957). Approximation to bayes risk in repeated play. *Annals of Mathematics Studies*, *39*, 97–139.

Hart, S., & Mas-Colell, A. (2000). A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, *68*, 1127–1150.

Hart, S., & Mas-Colell, A. (2001). A general class of adaptive strategies. *Journal of Economic Theory*, *98*, 26–54.

Hastie, T., Tibishirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer.

Herbster, M., & Warmuth, M. K. (2001). Tracking the best linear predictor. *Journal of Machine Learning Research*, *1*, 281–309.

Kalai, A., & Vempala, S. (2002). *Geometric algorithms for online optimization* (Technical Report). MIT.

Kivinen, J., & Warmuth, M. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, *132*, 1–64.

Kivinen, J., & Warmuth, M. (2001). Relative loss bounds for multidimensional regression problems. *Machine Learning Journal*, *45*, 301–329.

Littlestone, N., & Warmuth, M. K. (1989). The weighted majority algorithm. *Proceedings of the Second Annual Conference on Computational Learning Theory*.

Mahony, R., & Williamson, R. (2001). Prior knowledge and preferential structures in gradient descent algorithms. *Journal of Machine Learning Research*, *1*, 311–355.

Singh, S., Kearns, M., & Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. *Proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence* (pp. 541–548).

Zinkevich, M. (2003). *Online convex programming and generalized infinitesimal gradient ascent* (Technical Report CMU-CS-03-110). CMU.