Lecture 21: Competitive Analysis

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Motivation: Online Problems

Many problems in both finance and computer science reduce to trying to predict the future. . .

Examples from computer science include cache and virtual memory management.

Examples from finance typically revolve around predicting future returns for an asset, or designing a portfolio to maximize future returns.

Such problems become trivial if we know the future (i.e. the stream of future memory requests or tomorrow's newspaper), but typically we only have access to the past.

On Line vs. Off Line

An off-line problem provides access to all the relevant information to compute a result.

An *online problem* continually produces new input and requires answers in response.

Competitive Analysis

How can we theoretically evaluate how well an algorithm forecasts the future?

Statistical forecasts provide a predict the future that makes some sense in practice. However, they offer no future guarantees, particularly if the data distribution changes.

Competitive analysis offers a *worst-case* measure of the quality of the behavior of an algorithm which predicts the future.

We seek to compare the performance of algorithm A with only knowledge of the past with an algorithm which has complete knowledge of past and future makes *optimal* use of it.

Competitive Ratio

We say an online algorithm ALG is c-*competitive* if there is a constant α such that for all finite input sequences I,

$$ALG(I) \leq c \cdot OPT(I) + \alpha$$

Note that the additive constant α is a fixed cost that becomes unimportant as the size of the problem increases.
We do not particularly care about the run-time efficiency of ALG (except maybe that it is polynomial), but we do care about its competitive ratio c.

The Ski Rental Problem

Consider the problem of deciding when to purchase skis.
Whenever you go skiing, you can either rent skis for the day at cost x, or buy them for $b \cdot x$.
If you buy them the first day, the worst case is you never ski again, and you spent b times the optimal decision of simply renting.
Suppose you never buy them. After $k > b$ days, you have spent k/b times the optimal decision of buying from the first day.

When to Stop Renting...

But suppose you buy them after renting b times.

You did the right thing if you go $k < b$ times. If you go exactly b times, you spent twice as much as the optimal decision, but never changes after that.

Thus this "balancing" algorithm is 2-competitive.

We can view any online algorithm as a game between an online player (the skier) and a malicious adversary (his/her anterior cruciate ligament).

Searching for a Price

Suppose that we want to sell a indivisible asset (say a house) sometime over the next n days.

Say the price fluctuates on a daily basis in the real interval $[m, M]$, where m is the lowest possible price and M is the highest possible price.

What strategy can we use to sell the asset and get the highest possible price?

Buy Low, Sell High

If we knew the future history, the optimal strategy would be to sell at the *maximum price* occurring over the n days.

We seek a strategy which optimizes competitive ratio, i.e. which *minimizes* the *maximum ratio* of the price we get over the maximum price.

We do not seek a price which is good related to the "average", but good in the worst case.

What can we do?

Deterministic Price Searching

Note that if the price was high at one point but we didn't sell, our adversary could immediately and permanently lower the price to m.

Note that once we sell, our adversary can immediately raise the price to M.

At the end of the time period, we can always get a price of at least m.

Together, this suggests that we should sell the instant the price reaches some p^* which is high enough that we do OK in each instance.

The worst we do in the first case is p^*/m. The worst we do in the second case is M/p^*. Balancing them yields:

$$\frac{p^*}{m} = \frac{M}{p^*} \rightarrow p^* = \sqrt{Mm}$$

The *reservation price policy* (RPP) accepts the first price greater than or equal to $p^* = \sqrt{Mm}$.
Let $\phi = M/m$ define the *global fluctuation ratio*.
The competitive ratio c we get is

$$c = \sqrt{Mm}/m = \frac{\sqrt{M}}{\sqrt{m}} = \sqrt{\phi}$$

This is the optimal *deterministic* strategy.

Randomized Algorithms

Randomized algorithms use random numbers to design algorithms unlikely to encounter the worst possible case.

Randomization is particularly useful to make things difficult for an adversary to design a future that is bad for you.

We assume he has access to your program but not to read or effect the random numbers.

In an analysis of a randomized algorithms, we determine the expected value over all random number sequences for the *worst* possible input.

Thus our analysis is completely independent of the input distribution – randomized expectation has nothing to do with statistical expectation.

Example: Finding a Tire Leak

Suppose I ride over a tack and get a flat tire.

What is the best way I can find the location of the tack, assuming I can only explore $1/n$th of the wheel at a time?

If I use the *deterministic* strategy of repeatedly turning the wheel left $2\pi/n$ radians until I find the tack, my adversary can put the tack just right of the initial position.

This strategy yields a cost of n versus the optimal off-line cost of 1, for a competitive ratio of n.

Suppose instead I spin the wheel around randomly and then start walking to the left. Regardless of where the adversary puts the tack, my expected search cost (and competitive ratio) is $n/2$.

Randomized Price Searching

Suppose that $\phi = M/m = 2^k$ for some integer k, i.e. $M = m2^k$.

Let RPP_i be the deterministic *reservation price policy* where we sell as soon at the price hits $m2^i$.

The strategy $EXPO$ picks an integer i uniformly at random from $1, \ldots, k$, and then sells as the price hits $m2^i$.

There must be some j such that the optimum off-line return $p_{\max}$ lies between $m2^j \leq p_{\max} < m2^{j+1}$.

What can the adversary do? Knowing we are restricted to picking values of the form $m2^i$, they will pick a value of the form $m2^{j+1} - \epsilon$ to frustrate us for some j.

Analysis Idea

Suppose the adversary picks j.
Our target i was too big with probability $(k-j)/k$ – if so we were forced to settle for a price of m.
Our target i was less than $p_{\max}$ otherwise, meaning were able to realize our price.

Analysis

Our expected price is:

$$
\begin{aligned}
EXPO(j) &= \frac{k-j}{k}m + \frac{1}{k}\sum_{i=i}^{j} m2^i && (1)\\
&= \frac{m}{k}\left(k - j + \sum_{i=i}^{j} 2^i\right) && (2)\\
&= \frac{m}{k}\left(k - j + 2^{j+1} - 2\right) && (3)
\end{aligned}
$$

The competitive ratio is

$$
\begin{aligned}
c &= \frac{OPT}{EXPO(j)} = \frac{m2^{j+1} - \epsilon}{\frac{m}{k}\left(k - j + 2^{j+1} - 2\right)} && (4)\\
&\approx k\frac{2^{j+1}}{k - j + 2^{j+1} - 2} \rightarrow O(k) && (5)
\end{aligned}
$$

The competitive ratio is maximized when k is largest, for a competitive ratio of $O(\log \phi)$.

Note that the finer gradations in price level are at the low end of the range, not the high end where they mean more in absolute dollars.