# 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.

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.

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

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

Note that the additive constant $\alpha$ 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.

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?

If we knew the future history, the optimal strategy would be to that 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 buy 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 help them make decisions.

Randomization is particularly useful to make things difficult for an adversary – we assume the adversary has access to your program to design a future that is bad for you, but does not have access 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.

Randomization can be used to design simple algorithms which make it unlikely to encounter the worst possible case.

# 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?

Suppose I use the *deterministic* strategy of repeatedly turning the wheel left $2\pi/n$ radians until I find the tack.

But my adversary can put the tack just to the right of the initial position.

Then 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 buy soon at the price hits $m2^i$.

The strategy $EXPO$ picks an integer $i$ uniformly at random from $1, \ldots, k$, and then performs buys soon iff 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 $j$.

Suppose the adversary picks $j$.

Our target $i$ was too big with probability $(k-j)/j$ – 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

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

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

# One-Way Trading

A generalization of the price searching problem is to sell my entire assets over the trading period, but to remove the constraint that I must sell it all at once.

Suppose I am trying to liquidate my position in a stock. I may be able to better optimize my expected performance by using this freedom.

This is particularly true in real markets, as my sales serve to depress prices by increasing supply.

For this problem, it turns out that there is no difference between what competitive ratio is achievable with and without randomization.

What is a reasonable strategy?

# Threat-Based Strategies

Suppose we know that a competitive ratio of $c$ can be obtained.

If the current price is high and I don't sell, my adversary can drop it to $m$ and keep it there the rest of the period.

But if I do buy, the adversary can jack the price to $M$ at least momentarily, and I will be in trouble if I have already sold everything.

The *threat-based strategy* buys only when the price hits a new maximum. It buys just enough to ensure that we achieve a competitive ratio of $c$ if the price drops to $m$ for the rest of the game.

Analysis is needed to determine the optimal $c$ value and also how much to buy in response to price changes.

Clearly, we can achieve $c = O(\lg \phi)$, since we can use the randomized strategy and sell all at once.

Note that we can simulate the randomized strategy deterministically by putting $1/k$th of our money on each value of $i$.

The optimal threat-based strategy for one-way trading achieves a competitive ratio of $1/\ln 2 \approx 1.44$ times better the search bound of $EXPO$.

# Assessing the Model

How useful are our competitive algorithms for price searching and one-way trading?

Our assumptions of upper and lower bounds on possible price movements seem suspicious, although one can probably make save over/under-estimates of possible movements over short periods of time using historical data.

Still, wider upper/lower price bounds imply worse competitive ratios.

Guaranteeing you do, say half, as well as optimal doesn't look so good when the difference between the best investor and an index fund is often only a few percentage points.

That said, the randomized and threat-based heuristics seem to suggest reasonable approaches for one-way trading.

# Online Portfolio Selection

Suppose we can invest in a market with $s$ types of assets, including cash.

How should partition our money among the assets, and how should we adjust our portfolio to changes in asset prices?

The *buy and hold* strategy (BAH) strategy does not attempt to modify the portfolio for long periods of time.

*Buy and hold* results in very low transaction costs, and is historically better for individual investors to do than *market-timing* strategies which switch among investments seeking the best return.

Of course, market timing can yield much better returns *if* you do it right. Consider a stock that alternates returns of $d$ and $1/d$. Buy and hold returns at most $d$ over any period of length $n$, while the optimal market timing would yield a return of $d^{n/2}$.

A *constant rebalancing algorithm* always puts $1/s$ of our current wealth in each of the $s$ securities.

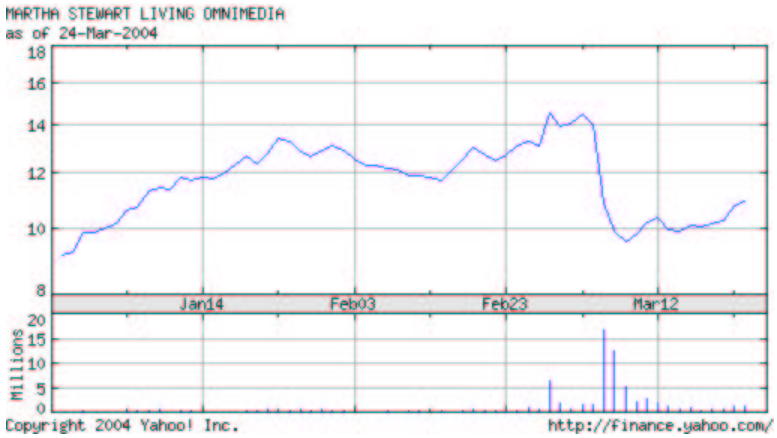
Such a diversified strategy enables us to pickup exponential growth over the previous return sequence if it starts positively.

Implementing a constant rebalancing strategy requires daily trades, but provides a way to capitalize on boom periods.

# Two-Way Trading

*Two-way trading* is a special case of online portfolio selection where you have only cash and one other security you can hold.

It differs from one-way trading in that we can shift back and forth between the two assets.




The optimal *offline* strategy is clear: put all your assets into the security on any day it offers positive returns. Otherwise, put everything into cash.

A trading strategy is said to be *money making* if it returns positive profit on every market sequence for which the optimal offline algorithm makes a profit.

The general adversary can ensure that *no* money making strategy exists.

If you are not initially invested in the non-cash asset, the next period will be the only one offering positive returns. If you are initially invested in the non-cash asset, offer such a negative return as to essentially wipe it out, then have a small enough positive return that you cannot recoup what you lost.

# The $(n, \phi)$ Adversary

Money making strategies are only possible against weaker adversaries.

An $(n, \phi)$ trading sequence is a $n - 1$ day sequence of returns for which the optimal offline strategy generates a return of at least $\phi$.

$$\phi = \prod_{i=1}^{n-1} \max\{1, r_i\}$$

An $(n, \phi)$ adversary is constrained to produce $(n, \phi)$ trading sequences.

We assume that you are given $n$ and $\phi$ in advance to help you plan your trading strategy.

Can you devise a provably money making strategy against an $(n, \phi)$ adversary?

How about when $n = 2$?

How about when $n = 3$?

# Making Money from the Adversary

For $n = 2$, we have only one trading period. Since this must produce a profit $\phi$, we should be fully invested in the non-cash asset.

For $n = 3$, we must look ahead to the case of $n = 2$. If we initially bet only on cash, the adversary will make that the only period of positive return.

If we are initially out of cash, the adversary can wipe us out now and show a $\phi$ in the next period.

We must bet something but not everything on the non-cash asset in the first round. If it shows positive return, we can quite the game with our holdings. If it shows negative return, we still have money and know there must be a positive return of at least $\phi$ in the remaining time.

Through such reasoning, for large $n$ we can work backwards from $n - 1$ to figure out the best first move to make.

# The Money Making Strategy

If $n = 2$, invest in the non-cash asset for return $R_2(\phi)$.

Otherwise, invest the fraction $b$ of your wealth in the non-cash asset, where

$$b = argmax_{b=0}^{1} \inf_{x \leq \phi} \{(bx + (1-b))R_{n-1}(\phi_{n-1})\}$$

$argmax$ returns the $b$ which maximizes the value, as opposed to max which returns the value.

Once you pick a $b$, your adversary will pick a return $x$ such as to minimize your wealth.

Your wealth after this event is your initial wealth times the returns on the cash and non-cash portions, i.e. $(bx + (1-b))$.

*After* the return $x$ is revealed to you, you can figure out the guaranteed return for the remaining period:

$$\phi_{n-j-1} = \min\{\phi_{n-j}, \phi_{n-j}/r_{j+1}\}$$

The best value of $b$ can be determined by dynamic programming.

Although this strategy is provably money-making, its can be yield poor returns for large $n$,

$$R_n(\phi) \leq \frac{1}{1 - (1 - 1/\phi)^{n-1}}$$

For large $n$, it can initially only afford to put a small amount of money into non-cash assets.

Since the optimal offline return is $\phi$, we get a not-inspiring competitive ratio of $\geq \max\{n-1, \phi\}$.