8 Online Algorithms for Search and Trading

8.1 Introduction to online algorithms

8.1.1 The Problem Type of Interest

- Input: A sequence of requests for service that may involve the use of some limited resources. The requests may be arriving over time and/or need to be processed in the order given in the sequence.
- Feasible solution: A sequence of actions/decisions on the requests satisfying some pre-specified properties. There are usually a large number of feasible solutions for a given input sequence.
- Objective function: $f(S)$ for feasible solution S that reflects quality of service (QoS).
- Optimal solution: A feasible solution S^* for which $f(S^*) = \min_{\forall S}\{f(S)\}$ in an minimization problem. (min is replaced by max in maximization problems.)
- Examples:
  - Bin Packing:
    Instance: $\sigma = (a_1, \ldots, a_n)$, where $a_i \in (0, 1]$.
    Goal: Pack the items in σ into the minimum number of unit-capacity bins.
  - Load Balancing:
    Instance: A sequence σ of n independent jobs $J_1, \ldots, J_n$, where p_j is the processing time of J_j.
    Goal: Schedule the jobs on m identical processors to minimize the makespan of the schedule.

8.1.2 Online and Offline Algorithms

- Online algorithm: Receives the requests one by one and performs an immediate action in response to each request without the knowledge of future requests.
- Offline algorithm: Receives the entire sequence in advance and makes a decision in response to each request based on the entire sequence.
- Online algorithms are more realistic approaches than offline algorithms but they often perform worse than their offline counterparts.
- Although an online algorithm may be optimal in that it produces a solution that optimizes the objective function, it is often approximate because of its lack of knowledge about future input.
- Examples:
  - The First-Fit algorithm (FF) for Bin Packing is online while the First-Fit-Decreasing algorithm (FFD) is offline.
  - The List Scheduling greedy algorithm (LS or GREEDY) for Load Balancing is online while the Longest Processing Time algorithm (LPT) is offline.

8.1.3 Competitive Analysis

- Let A be a online algorithm under study and OPT be the optimal offline algorithm.
- For any instance I, let $A(I)$ and $OPT(I)$ be the values of the objective function when applied to the solution constructed by A and the optimal solution, respectively. For a minimization problem, $A(I) \geq OPT(I)$.
- We say that A is c-competitive (or c is the competitive ratio of A) if $A(I) \leq c \cdot OPT(I) + a$ for any instance I and constant a. A is strictly c-competitive if $a = 0$.
- The equal part of $\leq$ means that there is at least one instance for which the equality holds, in which case we also say that the bound is tight.

- The ski rental problem: The input is n ski trips. For each trip, decide whether to rent (with cost of 1), buy (with cost of s), or use the skis already bought (with cost of 0). Consider a class of online algorithms $A(k)$ which don't know n and will rent for the first k trips and then buy, assuming $n \geq k+1$. The total cost of $A(k)$ is $k+s$. The optimal offline algorithm knows n and will incur a total cost of $\min\{n,s\}$. Considering the ratio of the two costs, we have $(k+s)/\min\{n,s\} \leq (k+s)/\min\{k+1,s\}$, which is minimized when $k+1=s$. Thus, the minimum ratio is $((s-1)+s)/\min\{s,s\} = (2s-1)/s = 2-(1/s)$. Therefore, the online algorithm that rents for the first $s-1$ trips and then buy has a competitive ratio of less than 2.

8.2 Introduction to financial problems

- Financial problems are attractive candidates for competitive analysis. Such problems are typical online in nature in that information about the future is often scarce and unreliable. The goal of most financial problems is to maximize the return or profit.

- Search: An online player is searching for the maximum (to sell) or the minimum (to buy) price in the market to trade his entire wealth all at once.

  An input sequence of prices of an investment option (which can be a stock, a fund, or a foreign currency), $\sigma = \{p_1, p_2, \ldots\}$, is given out one price value at a time. When a price pops out in a trading period, you (the online player) have to decide whether to accept the price to trade all of your wealth or wait for the next price that will be given in the following period.

  Applications of this problem include job and employee search.

- One-way trading: The player is searching for a series of prices to trade his wealth little by little in parts.

  σ is defined as in the search problem above. In addition, D_0 is given as the initial amount of wealth. The goal is to generate a sequence $D_1, D_2, \ldots$ with $D_0 \geq D_1 \geq D_2 \geq \cdots \geq 0$ such that $p_1(D_0 - D_1) + p_2(D_2 - D_1) + \cdots$ is maximized.

  Search is a special case of one-way trading, where $D_0 = \cdots = D_j$ and $D_{j+1} = 0$.

- Portfolio selection: The player wishes to periodically reallocate his wealth among a number of investment options.

- Relations between search and one-way trading:

  - Any randomized one-way trading algorithm is equivalent, in terms of expected returns, to a randomized search algorithm that trades the entire wealth at once at some randomly chosen period.
  - Any randomized search algorithm is equivalent to a deterministic one-way trading algorithm that trades the initial wealth in parts.
  - Combining the above two statements, we get that any one-way trading algorithm, deterministic or randomized, can be interpreted as a randomize search algorithm, and vice versa.

8.3 Competitive search algorithms

- Assumptions:

  - Each trading period is discrete, during which a price will be given. Le p_i be the price in the ith trading period, for $i = 1, \ldots, n$.
  - The number of trading periods, n, may be known or unknown to the online player. In the known duration case, the online player knows n in advance, In the unknown duration case, the player is informed that the game will end immediately before the final (nth) period begins.
  - Let $[m, M]$ denote the price range, i.e., $m \leq p_i \leq M$, for $i = 1, \ldots, n$. Assume that m and M are known by the online player. Define $\phi = M/m$ to be the global fluctuation ratio.
  - The player can always end the game by trading his asset at a price of at least m.
  - Here we consider the version of the search problem in which the online player searches for the maximum price to sell his asset,

- Reservation-Price-Policy (RPP): A deterministic search algorithm that accepts the first price greater than or equal to $p^* = \sqrt{Mm}$. Here p^* is called the reservation price.

  Theorem: RPP is $\sqrt{\phi}$-competitive.

  Proof: Let $p_{max} = \max\{p_1, \ldots, p_n\}$. Then the optimal price is p_{max}. Consider two cases.

  If $p_{max} \geq p^*$, then the ratio between the optimal price and the RPP price is at most $p_{max}/p* \leq M/p^*$.

  If $p_{max} < p^*$, RPP has to keep waiting until the final trading period when it has to accept p_n. So the ratio between the optimal price and the RPP price is $p_{max}/p_n \leq p^*/m$.

  Combining the two cases, the competitive ratio of RPP is at most $\max\{M/p^*, p^*/m\} = \sqrt{M/m} = \sqrt{\phi}$.

  Theorem: If only ϕ is known, no competitive ratio smaller than the trivial ϕ is achievable by any deterministic algorithm for the search problem.

- EXPO: A randomized search algorithm that with probability $1/k$, chooses algorithm RPP_i, $i = 1, \ldots, k$.

  Here we assume $\phi = 2^k$ for some integer k. We define RPP_i to be RPP with reservation price $m2^i$. So EXPO is in fact a uniform probability mixture over $\{\text{RPP}_i\}$.

  Theorem: EXPO is $c(\phi)\log\phi$-competitive, with $c(\phi)$ approaching 1 when $\phi \to \infty$.

  Proof: Let $p_{max} = \max\{p_1, \ldots, p_n\}$. Let j be an integer such that $m2^j \leq p_{max} < m2^{j+1}$. (Note that $m2^j \leq p_{max} \leq M = m\phi = m2^k$, so $j \leq k$.) Clearly, the optimal offline return is p_{max}.

  The particular choice of the interval $[m2^j, m2^{j+1})$ and the exact value of p_{max} are controlled by the adversary. It is advantageous to the adversary to choose p_{max} arbitrarily close to $m2^{j+1}$ because it increases the offline line return but does not change the online return at all, thus making the competitive ratio large. Hence, let $p_{max} = m2^{j+1} - \varepsilon$ for arbitrarily small $\varepsilon > 0$.

  For each $i \leq j$, $m2^i \leq m2^j < p_{max}$, so RPP_i accepts the reservation price $m2^i$. For each $i \geq j+1$, $m2^i \geq m2^{j+1} > p_{max}$, so RPP_i accepts the last price $p_n \geq m$. So on average, EXPO will return a price of at least

  $$\sum_{i=1}^{j}(\frac{1}{k}m2^i) + \sum_{i=j+1}^{k}(\frac{1}{k}m) = \frac{m}{k}(\sum_{i=1}^{j}2^i + (k-j)) = \frac{m}{k}(2^{j+1} + k - j - 2).$$

  For each particular choice of j, let $R(j)$ be the offline to online ratio. Thus, ignoring ε,

  $$R(j) = \frac{p_{max}}{\frac{m}{k}(2^{j+1}+k-j-2)} = \frac{m2^{j+1}}{\frac{m}{k}(2^{j+1}+k-j-2)} = \frac{2^{j+1}}{2^{j+1}+k-j-2}k = \frac{1}{1+\frac{k-j-2}{2^{j+1}}}k.$$

  Since the competitive ratio is the maximum ratio over all j, we determine that $R(j)$ is maximized at $j^* = k - 2 + \frac{1}{\ln 2}$. So the competitive ratio is EXPO is

  $$R(j^*)\frac{1}{1-\frac{\frac{1}{\ln 2}}{2^{k-1+\frac{1}{\ln 2}}}}k = c(\phi)\log\phi,$$

  where $c(\phi)$ approaches 1 as k (thus ϕ) grows. (Note that $k = \log\phi$.)

  Remarks:

  - Even if ϕ is not a power of 2, the above analysis can be extended and the competitive ratio still holds.
  - Even if the player does not know the values of m and M and knows only the global fluctuation ratio ϕ, the above competitive ratio still holds. In this case, however, RPP_i uses the reservation price of $p_1 2^i$, instead of $m2^i$.

- EXPO'_μ: A modified version of EXPO that, with probability $q(i)$, chooses algorithm RPP_i with reservation price $p_1 2^i$ for $i = 0, 1, \ldots$, where $q(i)$ is the probability of choosing i and $\mu = \{q(i) | i = 0, 1 \ldots\}$ is a probability distribution over all natural numbers.

8.4 Competitive one-way trading

- Recall that one-way trading trades the wealth not all at once, but in parts, with the goal to maximize the total return. Assume in the one-way trading problem discussed below, we wish to convert all our wealth in dollars to yens.
- A threat-based algorithm follows two rules:
  - Rule 1: Accept the current exchange rate (price) to trade only when it is the highest seen thus far.
  - Rule 2: To determine the amount of dollars to convert, only convert just enough to ensure that a competitive ratio c would be attained if an adversary dropped the exchange rate to the minimum possible rate in the next trading period and kept it there throughout the game.

    If m is known to the trade, the "minimum possible rate" will be m. If only ϕ is known and p is the highest rate seen so far, the "minimum possible rate" will be p/ϕ.

    What is the so-called threat? That the exchange rate will drop permanently to the minimum possible rate.
- Assumptions:
  - n, m, M are known,
  - THREAT refers to the optimal threat-base algorithm for one-way trading, with c^* to be its optimal attainable competitive ratio.
  - Since any price that is not a global maximum at the time it is revealed to the trader is rejected/ignored according to Rule 1, we can assume, WLOG, that in the input sequence, $p_1 < \cdots < p_k$ for some $k \leq n$. On the other hand, for the adversary to realize a threat, it will choose $k < n$ and $p_{k+1} = \cdots = p_n = m$.
- Notation:
  - D_i: The amount of remaining dollars immediately after the ith period, for $i = 0, \ldots, n$. For simplicity, assume $D_0 = 1$.
  - Y_i: The amount of accumulated yens immediately after the ith period, for $i = 0, \ldots, n$. Clearly, $Y_0 = 0$.
  - $s_i = D_{i-1} - D_i$: The amount in dollars to trade in the ith period, for $i = 1, \ldots, n$. Thus, $Y_i = Y_{i-1} + s_i \cdot p_i = \sum_{j=1}^{i} s_j \cdot p_j$.
- How to compute s_i, assuming c^* is known:

  In the ith period, for $i = 1, \ldots, n$, the price sequence is believed to be $p_1 < \cdots < p_i$ followed by $p_{i+1} = \cdots = p_n = m$ according to the threat. So the optimal return for this scenario is $1 \cdot p_i = p_i$ and the return of the THREAT algorithm is $Y_i + m \cdot D_i = Y_{i-1} + s_i \cdot p_i + m(D_{i-1} - s_i)$. The resulting ratio of the optimal return over the return by THREAT is no larger than c^*, so

$$\frac{p_i}{Y_i + mD_i} = \frac{p_i}{Y_{i-1} + s_i p_i + m(D_{i-1} - s_i)} \leq c^*.$$

  Since the smallest possible value of s_i is obtained when the above inequality becomes an equation, we then have

$$\frac{p_i}{Y_i + mD_i} = \frac{p_i}{Y_{i-1} + s_i p_i + m(D_{i-1} - s_i)} = c^*.$$

  Thus, we get

$$s_i = \frac{p_i - c^*(Y_{i-1} + mD_{i-1})}{c^*(p_i - m)} \text{ (Eq.1) and } \frac{p_i}{c^*} = Y_i + mD_i \text{ (Eq.2).}$$

  From Eq. 1 and $Y_0 = 0$, $D_0 = 1$, we get

$$s_1 = \frac{p_1 - c^* m}{c^*(p_1 - m)}.$$

  In the case of $i > 1$, replacing i with $i - 1$ in Eq. 2, we get $\frac{p_{i-1}}{c^*} = Y_{i-1} + mD_{i-1}$. So back to Eq. 1,

$$s_i = \frac{p_i - c^*(Y_{i-1} + mD_{i-1})}{c^*(p_i - m)} = \frac{p_i - c^* \frac{p_{i-1}}{c^*}}{c^*(p_i - m)} = \frac{p_i - p_{i-1}}{c^*(p_i - m)}.$$

- How to compute c^*:

  Recall that we use k to denote the largest period index for which $p_1 < \cdots < p_k$ and that after this period the price drops down to m. The best online algorithm should spread out the trading of its wealth (\$1) throughout the first k trading period and trade its final portion in the kth period. Therefore, $\sum_{i=1}^{k} s_i = 1$. So

$$1 = \sum_{i=1}^{k} s_i = s_1 + \sum_{i=2}^{k} s_i = \frac{p_1 - c^* m}{c^*(p_1 - m)} + \sum_{i=2}^{k} \frac{p_i - p_{i-1}}{c^*(p_i - m)}.$$

  Solving it for c^*, we get

$$c^* \equiv c^*(k, m, p_1, \ldots, p_k) = 1 + \frac{p_1 - m}{p_1} \sum_{i=2}^{k} \frac{p_i - p_{i-1}}{p_i - m}.$$

  We use $c_n^*(m, M)$ to denote the optimal (worst-case) competitive ratio for the n-period game. Thus,

$$c_n^*(m, M) = \max_{k \le n \text{ and } m \le p_1 < \cdots < p_k \le M} c^*(k, m, p_1, \ldots, p_k).$$

  Unfortunately, an explicit expression for $c^*(m, M)$ cannot be obtained.