Online Bidding

Hsiang-Hsuan (Alison) Liu

Consider the following game played by Sandy and Bill. Sandy have a positive integer *target* in mind without telling Bill the value of target. Bill wants to know the value of target by making a sequence of guesses. In each round, Bill proposes a potential number, and Sandy tells him *higher* if the proposed number is less than the target. In this case, Bill proceeds to the next round and proposes the next guess. Otherwise, if the proposed number is equal to or grater than target, Sandy tells Bill that he *hits* the target and the game is over. To make the game more interesting, Bill aims to hit the target with the sum of the proposed values to be as small as possible.

This game is called *(online) bidding*. It is a simple abstract model of many auction or similar decision making scenarios. The game can be viewed as an online problem. The target which is unknown to the algorithm is an uncertain factor. The algorithm needs to make decisions, which is the next guess, without knowing what the target is. If the target is known to the player, then the player can achiever optimal cost by proposing exactly the target. However, as the target is hidden from the player, the adversary (that is, the ONE who chooses the target) can select a target that is the worst case of the player's strategy.

OnlineBidding problem

The online algorithm (player) wants to guess the target number $t \geq 1$ which is unknown to it. To this end, the player submits a sequence $B = (b_1, b_2, \cdots)$ of positive *bids* where $b_i > b_{i-1}$ for all $i > 1$, until one of them with value higher than or equal to t. The cost of the strategy of a player is defined by this sequence of bids, $\sum_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.

1 Doubling: A 4-competitive online algorithm

Algorithm 1 A 4-competitive online algorithm for the OnlineBidding problem

```
1: Set the first bid to be 1
2: while The latest bid misses do
3:     Set the next bid to twice the last one
```

Theorem 1. *Algorithm 1 is at least 4-competitive when the target is large enough.*

Proof. Consider the adversary where $t = 2^k + 1$ for a large enough k. Algorithm 1 pays $1 + 2 + 4 + \cdots + 2^k + 2^{k+1} \approx 2 \cdot 2^{k+1}$ (since k is very large). The ratio

$$\frac{\text{ALG}(t = 2^k + 1)}{\text{OPT}(t = 2^k + 1)} = \frac{2 \cdot 2^{k+1}}{2^k + 1} = \frac{4 \cdot (2^k + 1) - 4}{2^k + 1} = 4 - \frac{4}{2^k + 1}.$$

When k is very large, the lower bound tends to 4. □

Theorem 2. *Algorithm 1 attains a competitive ratio 4.*

Proof. We prove this theorem by showing that whatever t is, Algorithm 1 cost is at most $4t$.

According to Algorithm 1, $b_i = 2^{i-1}$ for $i \geq 1$. The competitive ratio of the algorithm is $\sum_{i=1}^{k} b_i / t$ where t is the target number and b_k is the smallest number in B that is at least t. The target number t

must be greater than b_{k-1}, otherwise b_{k-1} is the smallest number in B that is at least t instead of b_k. Without loss of generality, we assume $t = b_{k-1} + x$ where $x \geq 1$. The competitive ratio of Algorithm 1 is computed as follows.

$$\frac{\mathrm{ALG}(t)}{\mathrm{OPT}(t)} = \frac{\sum_{i=1}^{k} b_i}{t} = \frac{\sum_{i=1}^{k} b_i}{b_{k-1} + x} < \frac{\sum_{i=1}^{k} b_i}{b_{k-1}} = \frac{\sum_{i=1}^{k} 2^{i-1}}{2^{k-2}} = \frac{2^k - 1}{2^{k-2}} < \frac{2^k}{2^{k-2}} = 4.$$

The second last equality follows from the formula of geometric series.

□

Note that the analysis is nearly tight when the target is super large.

2 Doubling Algorithms

In some problems, the optimal cost increases as the instance is revealed. We For these problems, one can design online algorithms using the concept of *doubling* with a parameter r. The rough idea is that by using geometrically increasing estimation on the optimal solution, we can produce fragments of the algorithm's solution. Moreover, since we have a lower bound of the optimal cost, we can use this lower bound as a "budget" to design an online algorithm. As long as we can guarantee that the total cost of the online algorithm is at most c times the budget, this online algorithm has all freedom to do anything it needs while being c-competitive.

Doubling algorithm framework. Throughout the whole process (that is, the sequence of events releasing), we keep track of the value of the optimal cost OPT for the instance that is seen so far. Note that the optimal cost OPT is increasing as the instance is revealed further.

The doubling algorithm works in *phases*. The phases can be seen as a partition of the time horizon or a events sub-sequences. *Phase* i starts from the first time when $\mathrm{OPT} \geq r^i$. If we can show that for each phase i, the incurred (total) cost of our doubling algorithm so far never exceeds $c \cdot r^i$, the doubling algorithm is c-competitive. (See Figure 1.)

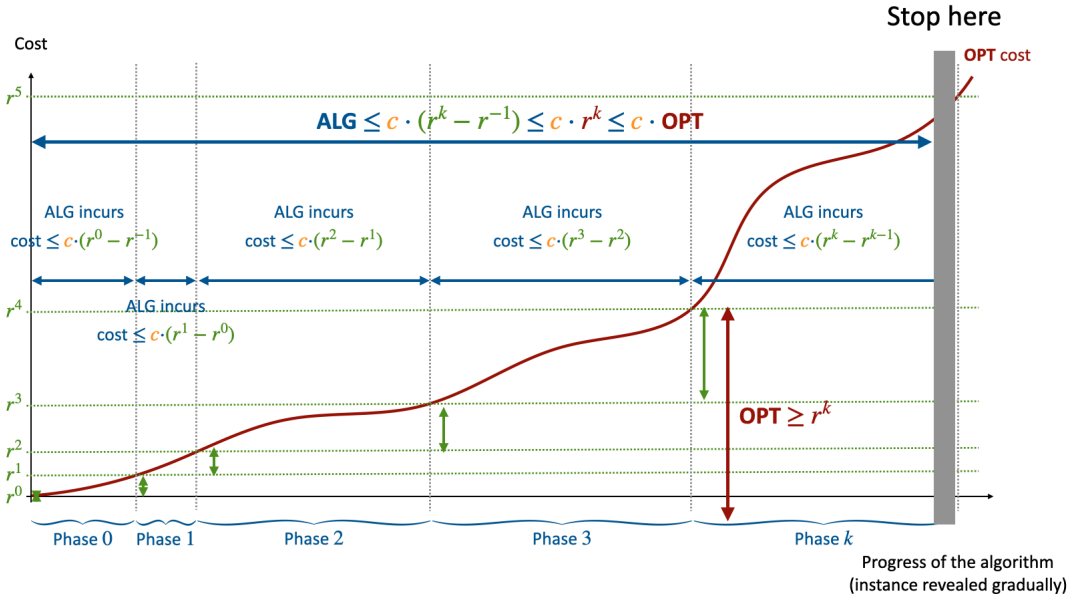

Figure 1: Illustration of the doubling framework

The parameter r is a factor for tracking the growth of the optimal cost. That is, the new phase starts at the time when OPT is increased to r times the optimal cost at the start of the previous phase. In many cases, we choose r to be 2, and that is where this term "doubling" comes from. However, sometimes we also use parameters other than 2.

Algorithm 1 is actually a doubling algorithm. Each bid of Algorithm 1 creates a phase. In phase 0, the algorithm proposed bid $1 = 2^0$. If it does not hit the target, we know that $\text{OPT} > 2^0 = 1$ and phase 1 starts. Similarly, in phase i where $i \geq 1$, if the bid $b_{i+1} = 2^i$ does not hit the target, we know $\text{OPT} > 2^i$. Then it starts the new phase $i+1$ and incurs a new cost 2^{i+1}.

In each phase k, Algorithm 1 incurs new cost of 2^k. Hence, the total cost of Algorithm 1 until phase k is $2^k + \sum_{i=0}^{k-1} 2^i \leq 2 \cdot 2^k$. As we know that at the beginning of phase k, $\text{OPT} > 2^{k-1}$, the total cost (so far) is at most $2 \cdot 2^k < 4 \cdot \text{OPT}$.

Although doubling is an useful framework for designing online algorithms, it is not always applicable. For some problems where the optimal cost does not necessarily increase as the instance reveals, doubling does not work.

3 A general lower bound for the OnlineBidding problem (It is complicate and optional)

Recall that the cost of any strategy B is $\sum_{i=1}^{k} b_i$ where b_k is the first number that is at least the target t. An idea for designing the adversary is to choose $t = b_{k-1} + 1$ such that at least one number b_k in B is potentially much larger than t. In this case, the competitive ratio of B is $\sum_{i=1}^{k} b_i/(b_{k-1}+1)$.

The adversary $t = b_{k-1} + 1$ benefits from an extremely large k. Recall that $b_i > b_{i-1}$ for all $i > 1$ and $b_1 \geq 1$. This implies $b_i \geq i$ for all i. Hence, when k tends to an arbitrarily large number, b_{k-1} is also arbitrarily large. In this case, the competitive ratio of B is arbitrarily close to $\sum_{i=1}^{k} b_i/b_{k-1}$.

To find the lower bound, we need the following proposition.

Proposition 1. *Consider $s \geq 1$ as a constant integer and $h > 1$ as a constant number. If $b_i/b_{i-1} \leq h$ for all $i > s$, then $\sum_{i=s}^{j-2} b_i \geq b_{j-1}/(h-1)$ when j tends to infinity.*

Proof. We find a lower bound for each term in $\sum_{i=s}^{j-2} b_i$ and then compute the sum of them. By the given condition, we have $b_{j-2} \geq b_{j-1}/h$. By the same condition, we have $b_{j-3} \geq b_{j-2}/h \geq b_{j-1}/h^2$. In general, $b_{j-x} \geq b_{j-1}/h^{x-1}$ for all $2 \leq x \leq j-s$. Consider the sum of these terms, we have the following bound.

$$\sum_{i=s}^{j-2} b_i = \sum_{x=2}^{j-s} b_{j-x} \geq \sum_{x=2}^{j-s} \frac{b_{j-1}}{h^{x-1}} = \frac{\frac{1}{h} - (\frac{1}{h})^{j-s}}{1-\frac{1}{h}} \cdot b_{j-1} = \frac{1-(\frac{1}{h})^{j-s-1}}{h-1} \cdot b_{j-1}.$$

The second equality follows from the formula of geometric series. When j tends to infinity, $j-s-1$ also tends to infinity. In this case, $(1/h)^{j-s-1}$ tends to 0 as $1/h < 1$. Thus the proposition follows. □

Theorem 3. *For the OnlineBidding problem, any deterministic online algorithm has a competitive ratio of at least 4.*

Proof. We divide all possible strategies into two types. Let $h > 1$ be the growth rate of b_k, i.e., $h = b_k/b_{k-1}$. For some arbitrarily large k, the types are defined as follows.

- The first type of strategies is that *all* growth rates before b_k are upper bounded by that of b_k, i.e. $b_i/b_{i-1} \leq h$ for all $i \leq k$.
- The other type of strategies is that some of the preceding growth rates are larger than that of b_k, i.e., there exists $s < k$ such that $b_s/b_{s-1} > h$.

We show that the competitive ratio of any strategy is at least 4 for these two types of strategies separately.

The first type of strategies is that for arbitrarily large k and for all $i \leq k$, $b_i/b_{i-1} \leq h$ where $h = b_k/b_{k-1}$. The competitive ratio of strategy B is as follows.

$$\frac{\sum_{i=1}^{k} b_i}{b_{k-1}+1} \approx \frac{\sum_{i=1}^{k} b_i}{b_{k-1}} = \frac{\sum_{i=1}^{k-2} b_i + b_{k-1} + b_k}{b_{k-1}} = \frac{\sum_{i=1}^{k-2} b_i}{b_{k-1}} + 1 + h \geq \frac{1}{h-1} + 1 + h.$$

The last inequality follows from Proposition 1. The formula $1/(h-1)+1+h$ has the minimum 4 when $h > 1$.

The second type of strategies is that for arbitrarily large k, there exists $s < k$ such that $b_s/b_{s-1} > h$ and b_s is selected as the largest number in B satisfying the condition. When k is large enough, by showing that B is convergent, we prove in the following that b_s/b_{s-1} can also be upper bounded by some function of h. Therefore, we can obtain a similar competitive ratio lower bound to that of the first type.

We consider two cases of s: s is a constant, or s grows as k grows. If s is a constant, which means that s does not depend on k, then we can remove $b_1, \cdots, b_{s-1}$ from B without decreasing the cost of B too much. After removing $b_1, \cdots, b_{s-1}$ from B, the new cost $\sum_{i=s}^{k} b_i/(b_{k-1}+1) \leq \sum_{i=1}^{k} b_i/(b_{k-1}+1)$. In this case, we can analyze the competitive ratio by the similar method to the analysis of the first type of strategies and by Proposition 1.

$$\frac{\sum_{i=s}^{k} b_i}{b_{k-1}+1} \approx \frac{\sum_{i=s}^{k-2} b_i}{b_{k-1}} + 1 + h \geq \frac{1}{h-1} + 1 + h \geq 4.$$

For the case where s grows as k grows and $b_s/b_{s-1} > b_k/b_{k-1}$, s can be made arbitrarily large by making k to be arbitrarily large. There must exist a number $s' < s$ such that $b_{s'}/b_{s'-1} > b_s/b_{s-1}$ and $b_{s'}$ is selected as the largest number in B satisfying the inequalities, otherwise s is chosen as k in the first type and B is categorized to the first type. Without loss of generality, we assume that s' grows as s grows and can be made arbitrarily large. In this case, we can find recursively $s'' < s'$ and so on. Let D be the index sequence containing those selected numbers, i.e., $D = (\cdots, s'', s', s, k)$. The sequence D contains infinite number of elements since k tends to infinity.

Let d_i be the i-th number of D and R be the rate sequence

$$\left(\frac{b_{d_1}}{b_{d_1-1}}, \frac{b_{d_2}}{b_{d_2-1}}, \cdots, \frac{b_{d_{|D|}}}{b_{d_{|D|}-1}} \right).$$

The sequence R has the same length with that of D and thus is infinitely long. By the selection of the numbers in D, the rate sequence R is decreasing. Since $b_i/b_{i-1} \geq 1$, R eventually converges to some fixed number. As k is the last number of D, R converges to $h = b_k/b_{k-1}$. When the length of R is large enough, there exists a number j such that for all $i \geq j$, $b_{d_i}/b_{d_i-1} \leq (1+\epsilon) \cdot h$ for some $\epsilon \geq 0$. By finding a sufficiently large k, ϵ tends to 0 and the number of considered growth rates $|D| - j$ tends to infinity. This implies that j can be treated as a constant with respect to a sufficiently large $|D|$ and thus d_j can also be treated as a constant with respect to a sufficiently large k. The competitive ratio of strategy B is as follows.

$$\begin{aligned}\frac{\sum_{i=1}^{k} b_i}{b_{k-1}+1} \approx \frac{\sum_{i=1}^{k} b_i}{b_{k-1}} \geq \frac{\sum_{i=d_j}^{k} b_i}{b_{k-1}} &= \frac{\sum_{i=d_j}^{k-2} b_i}{b_{k-1}} + 1 + h \\ &\geq \frac{1}{(1+\epsilon)\cdot h - 1} + 1 + h \approx \frac{1}{h-1} + 1 + h \geq 4.\end{aligned}$$

The second last inequality follows from Proposition 1 and $b_i/b_{i-1} \leq (1+\epsilon) \cdot h$ for all $d_j < i \leq k$. □