Introduction to Algorithms

Online Algorithm

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2021

Introduction

Online Algorithms

Online Algorithms are algorithms that need to make decisions without full knowledge of the input, i.e., with full knowledge of the past but no (or partial) knowledge of the future.

For this type of problem we will attempt to design algorithms that are *competitive* with the optimal offline algorithm, which has perfect knowledge of the future.

Competitive Ratio

- Competitive ratio: For the maximization problems,

$$ratio = \max_{s} \frac{ALG(s)}{Offline\ OPT(s)}$$

  , where $ALG(s)$ is the cost of ALG on the input sequence s and Offline $OPT(s)$ is the optimal cost for the same sequence with full information.
- Competitive ratio is a worst case bound.

The Ski-Rental Problem

- Assume that you are taking ski lessons. After each lesson you decide (depending on how much you enjoy it, what is your bones status, and the whether) whether to continue to ski or to stop totally.
- You have the choice of either renting skis for 1$ a time or buying skis for B$.
- Will you buy or rent?

The Ski-Rental Problem

- If you knew in advance how many times T you would ski in your life then the choice of whether to rent or buy is simple. If you will ski more than B times then buy before you start, otherwise always rent.
- The cost of this algorithm is min(T, B).
- This type of strategy, with perfect knowledge of the future, is known as an offline strategy.

The Ski-Rental Problem

- In practice, you don't know how many times you will ski. What should you do?
- An online strategy will be a number k such that after renting $k-1$ times you will buy skis (just before your k^{th} visit).

Claim:

Setting $k = B$ guarantees that you never pay more than twice the cost of the offline strategy.

Example: Assume $B = 7\$$ Thus, after 6 rents, you buy. Your total payment: $6+7 = 13\$$

The Ski-Rental Problem

Theorem:

Setting $k = y$ guarantees that you never pay more than twice the cost of the offline strategy.

Proof:

When you buy skis in your k^{th} visit, even if you quit right after this time, $T \geq B$.

- Your total payment is $\mathrm{k} - 1 + \mathrm{B} = 2\mathrm{B} - 1$.
- The offline cost is $\min(\mathrm{T}, \mathrm{B}) = \mathrm{B}$.
- The ratio is $(2B - 1)/B = 2 - 1/B$.

We say that this strategy is (2-1/B)-competitive.

The Ski-Rental Problem

Is there a better choice of k?

- Let k be any strategy (buy after **k-1** rents).
- Suppose you buy the skis at the k^{th} time and then break your leg and never ski again.
- Your total ski cost is $k-1+B$ and the optimum offline cost is **min(k,B)**.
- For every k, the ratio $(k-1+B)/min(k,B)$ is at least $(2-1/B)$.
- Therefore, every strategy is at least $(2-1/B)$- -competitive.

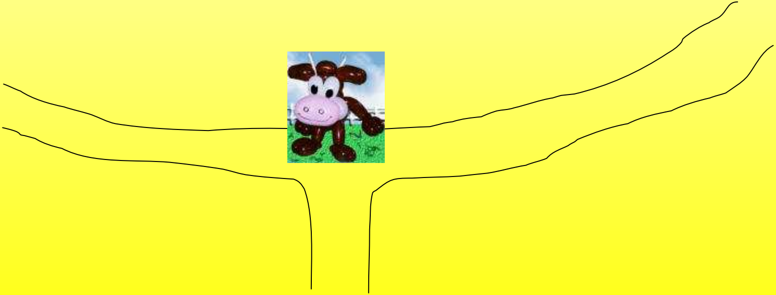
The Ski-Rental Problem

The general rule:

When balancing small incremental costs against a big one-time cost, you want to delay spending the big cost until you have accumulated roughly the same amount in small costs.

The Lost Cow Problem

Old McDonald lost his favorite cow. It was last seen marching towards a junction leading to two infinite roads. None of the witnesses can say if the cow picked the left or the right route.



The Lost Cow Problem

OLD MCDONALDS ALGORITHM()

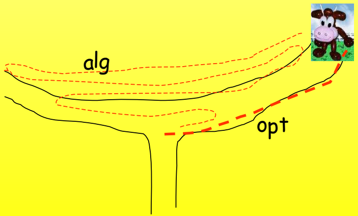
```
d = 1;current side = right
while true do
    Walk distance d on current side
    if find cow then
        exit
    else
        d = 2d
        Flip current side
        return to starting point
```

The Lost Cow Problem

Theorem

Old McDonald'algorithm is 9-competitive.

In other words: The distance that Old McDonald might pass before finding the cow is at most 9 times the distance of an optimal offline algorithm (who knows where the cow is.).



The Lost Cow Problem

Theorem

Old McDonald'algorithm is 9-competitive.

Proof:

The worst case is that he finds the cow a little bit beyond the distance he last searched on this side (why?) [1].

Thus, OPT $2^j + \varepsilon$ where $j = \#$ of iterations and ε is some small distance. Then,

$$\begin{aligned} Cost\ OPT &= 2^j + \varepsilon > 2^j \\ Cost\ ON &= 2(1+2+4+\ldots+2^{j+1}) + 2^j + \varepsilon \\ &= 2 \cdot 2^{j+2} + 2^j + \varepsilon = 9 \cdot 2^j + \varepsilon < 9 \cdot Cost\ OPT \end{aligned}$$

[1] *Note: this implies that at the first try, you search the direction where the cow is.*

The Secretary Problem

We have n candidates (perhaps applicants for a job or possible marriage partners). Our goal is choose the very best candidate. The assumptions are

- Candidates can be totally ordered from best to worst with no ties.
- Candidates arrive sequentially in random order.
- We can only determine the relative ranks of the candidates as they arrive. We cannot observe the absolute ranks.
- After each interview we must either immediately accept or reject the applicant. Once a candidate is rejected, she can not be recalled. Once a candidate is accepted, we stopped interviewing.
- The number of candidates n is known.

The Secretary Problem

An Online Strategy

- After meeting the i-th candidate, we are able to give a score denoted $\text{score}(i)$.
- Selecting a positive integer $k < n$, interviewing and then rejecting the first k candidates.
- Accept the first candidate thereafter who has a higher score than all k preceding candidates.
- If it turns out that the best-qualified candidate was among the first k interviewed, then we have to accept the n-th applicant.

The Secretary Problem

$\textsc{On-Line-Maximum}(k, n)$

```
bestscore = −inf
for i = 0 to k do
    if score(i) > bestscore then
        bestscore = score(i)
for i = k + 1 to n do
    if score(i) > bestscore then return i
return n
```

The Best Possible k

Let S be the event that we succeed in choosing the best-qualified candidate. We choose the k to maximize $Pr\{S\}$.
Let S_i be the event that we succeed when the best-qualified applicant is the i-th one interviewed. We assume k is fixed.

$$Pr\{S\} = \sum_{i=1}^{n} Pr\{S_i\} = \sum_{i=k+1}^{n} Pr\{S_i\}$$

Let B_i be the event that the best-qualified applicant must be in position i, and let O_i be the event that none of the applicants in positions from $k+1$ to $i-1$ are chosen. Then,

$$Pr\{S_i\} = Pr\{B_i \cap O_i\} = Pr\{B_i\}Pr\{O_i\}$$

The best possible k

The maximum score is equally likely to be in any one of the n positions.

$$Pr\{B_i\} = 1/n$$

For event O_i to occur, the maximum value in positions from 1 to $i-1$ must be in one of the first k positions.

$$Pr\{O_i\} = k/(i-1)$$

$$Pr\{S_i\} = Pr\{B_i\}Pr\{O_i\} = k/(n(i-1))$$

And we can calculate the possibility of S.

$$Pr\{S\} = \sum_{i=k+1}^{n} Pr\{S_i\} = \sum_{i=k+1}^{n} \frac{k}{n(i-1)} = \frac{k}{n}\sum_{i=k+1}^{n} \frac{1}{i-1} = \frac{k}{n}\sum_{i=k}^{n-1} \frac{1}{i}$$

The best possible k

We have

$$\int_k^n \frac{1}{x}\,dx \le \sum_{i=k}^{n-1} \frac{1}{i} \le \int_{k-1}^{n-1} \frac{1}{x}\,dx$$

Evaluating these definite integrals.

$$\frac{k}{n}(\ln n - \ln k) \le Pr\{S\} \le \frac{k}{n}(\ln(n-1) - \ln(k-1))$$

Choosing k that maximizes the lower bound on $Pr\{S\}$.

$$\frac{1}{n}(\ln n - \ln k - 1)) = 0 \to k = \frac{n}{e}$$

If we implement our strategy with $k = n/e$, we succeed in hiring our best-qualified applicant with probability at least $1/e$.