1 Introduction

In the 50’s and 60’s, algorithms for exact solutions were the main concern.

An optimization problem: given problem input $I$, find optimal solution $OPT(I)$.

Usually we want to minimize the time complexity of the algorithm.

An efficient algorithm should have polynomial time complexity: $O(|I|^c)$ for some constant, $c$.

Examples:
- Minimum Spanning Tree (MST)
- Shortest Paths
- Flow Problems

At the end of the 60’s, it was discovered that some families of problems seem to be unsolvable within these requirements.
These are the NP and NP-Complete problems.
E.g., the traveling salesman problem (TSP).

Definition 1. Algorithm $A$ (for a minimization problem) is an $\alpha$-approximation if $\forall I : A(I) \leq \alpha \cdot OPT(I)$

(The time complexity is usually polynomial.)

Obviously, smaller $\alpha$ implies better approximation.

Note that for maximization problem, the relation is inverted, so $A(I) \geq \alpha \cdot OPT(I)$ for $\alpha < 1$. (Sometimes $A(I) \geq \frac{1}{\alpha} OPT(I)$ for $\alpha > 1$)

Later a new type of algorithms were developed: Online Algorithms.

In this course we consider this type, where not all the information is given beforehand.

2 Examples

2.1 Ski Rental

At any time we can
- Buy for $M$
- Rent for another day for $1$
Although we have to decide each day, we shall look at it as a Continuous process. This problem is analogous to the **context switch problem**, where we have to decide whether to wait for input or to switch process. In both problems, we do not know beforehand for how long we may need the resource.

The offline version is trivial: “If we need it for more than M time, buy. Otherwise, rent”.

To elaborate:

Say we need it for *t* time-units.

If \( t < M \), rent, and \( \text{OPT} = t \).

If \( t \geq M \), buy, and \( \text{OPT} = M \).

### 2.2 Path Search

We are on the beach at night, with a flash-light. we can walk in one of two directions along the shore, in order to find a treasure. Only once we are over the treasure, we see it and win. The goal is to reach the treasure in a minimal length path, and the solution is the path description itself.

The offline version is trivial: “walk in the right direction up-to the treasure”.

### 2.3 Paging

Let’s assume two levels of memory:

- Main memory, which is made of *k* pages. Transfer rate: 300 million pages/second.
- Hard disk - external memory, made of infinite number of pages. Transfer rate: 300 pages/second (million times slower than main)

The input is a series of requests, e.g., 7,3,1,2,3,7,4,... Whenever a page is brought, an existing one is thrown out of the main memory. The goal is to minimize the number of page-faults.

In the offline version we know the request sequence in advance. The solution: “throw the page we will need in the most distant future”. In the online version, OSs use the LRU (Least Recently Used) algorithm, which is a “time-wise” reflection of the offline solution. That is, the past and future are interchanged.

### 2.4 Scheduling

We have *m* identical machines, and a series of tasks with specified load, \( w_1, w_2, w_3, ... \) As a task arrives, we have to assign it to a machine. So we have an assignment algorithm, \( A(i) = j \), where *i* is the task index and *j* is the machine index.

The **load of a machine** is the sum of the tasks loads: \( l_j = \sum_{i|A(i)=j} w_i \)

And the **max load** is: \( \max_j l_j \)

The **goal** is to minimize \( \max_j \text{ load} \).

The common practice is **Round Robin** algorithm, where every incoming task is given to the next machine in a circular fashion.
The offline version, even for \( m=2 \), is equivalent to the knapsack (or partition) problem, which is NP-Complete (in a “week” manner for 2 machines, and “strong” for larger \( m \).) It can be solved approximately within \( 1 + \epsilon \) for each \( \epsilon > 0 \), and the time would be, for example, \( n^{(1/\epsilon)^4} \), i.e., PTAS (Polynomial Time Approximation Scheme.)

**How do we assess online algorithms?**

**Ski Rental:** If we want to minimize the maximum payment, we buy immediately.

**Paging:** A page-fault is possible any minute for all algorithms - it is easy to give the worst-case scenario.

**Options for assessment:**

- Assume some distribution of the input, then use the expectation. If this is possible, it is not such a bad idea.
- Test algorithm for probable inputs. E.g, for Paging, test on historical traces. But history based forecast is not perfect, and some singular events may occur.
- Approximation ratio - Competitive ratio:

\[
\sup_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}
\]

As used in approximation algorithm, only here the input is not given in advance but is a series, \( \sigma \). \( OPT \) is the algorithm that knows all this series beforehand.

**Definition:** Algorithm \( ALG \) (for minimization problem) is \( C \)-Competitive if

\[
\forall \sigma : ALG(\sigma) \leq C \cdot OPT(\sigma)
\]

where the offline algorithm \( OPT \) knows all the input in advance.
3 Analysis

3.1 Ski Rental

ALG = time of buy
ON = online algorithm cost
OPT = optimal algorithm cost

<table>
<thead>
<tr>
<th>ALG</th>
<th>ON</th>
<th>OPT</th>
<th>Competitive Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M</td>
<td>min(t,M)</td>
<td>∞</td>
</tr>
<tr>
<td>M/2</td>
<td>( \begin{cases} t &amp; \text{if } t &lt; M/2 \ (3/2)M &amp; \text{otherwise} \end{cases} )</td>
<td>min(t,M)</td>
<td>3 (= ( \frac{(3/2)M}{M/2} ))</td>
</tr>
<tr>
<td>M</td>
<td>( \begin{cases} t &amp; \text{if } t &lt; M \ 2M &amp; \text{otherwise} \end{cases} )</td>
<td>min(t,M)</td>
<td>2 (= ( \frac{2M}{M} ))</td>
</tr>
<tr>
<td>∞</td>
<td>t</td>
<td>min(t,M)</td>
<td>∞ (as ( t \to \infty ))</td>
</tr>
<tr>
<td>2M</td>
<td>( \begin{cases} t &amp; \text{if } t &lt; 2M \ 3M &amp; \text{otherwise} \end{cases} )</td>
<td>min(t,M)</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>( \begin{cases} t &amp; \text{if } t &lt; 7 \ 7 + M &amp; \text{if } t \geq 7 \end{cases} )</td>
<td>min(t,M)</td>
<td>( \frac{7 + M}{7} ) (large for large ( M ))</td>
</tr>
</tbody>
</table>

Note: The latter algorithm is not a homogeneous case, so we can dismiss it.

Theorem 3.1. Algorithm that buys after time \( M \) is 2-Competitive.

Proof. Let \( t \) denote the time in the series.
If \( t < M \) then \( ON(\sigma) = OPT(\sigma) = t \)
Otherwise, \( t \geq M \), so \( ON(\sigma) = 2M \) and \( OPT(\sigma) = 2M \) and \( ON(\sigma) \leq 2OPT(\sigma) \)

Theorem 3.2. Every (deterministic) online algorithm for the Ski-Rental problem is at least 2-competitive. (Note that 2-competitive is also 5-competitive.)

Proof. For any given algorithm, we shall come-up with a series that it fails for. Lets take any (deterministic) algorithm, “buy after \( T \).” For \( T = \infty \), the competitive ratio is surely larger than any constant, in particular from 2. (We can take a series of length, say, 4M) Otherwise, we shall take a series \( \sigma \) of length \( T \) (specifically chosen for this algorithm.) \( ON(\sigma) = T + M \) and \( OPT(\sigma) = min(T, M) \) So for every \( T \):

\[
\frac{ON(\sigma)}{OPT(\sigma)} = \frac{T + M}{min(T, M)} = \frac{T}{min(T, M)} + \frac{M}{min(T, M)} \geq 1 + 1 = 2
\]
3.2 Path Search

Every finite move we make will be non-competitive since the treasure may be $10^7$ times closer on the opposite direction.

So we have two options:

- Assume that $OPT \geq 1$
- Introduce an additive constant, $\beta$.

And we pick the second option:

**Definition 2.** Algorithm is $C$-Competitive if there exists such a “global constant”, $\beta$, for which:

$$ON(\sigma) \leq C \cdot OPT(\sigma) + \beta$$

**Note:** If the first move is finite there is no competitive algorithm using the original definition.

**Solution 1: BFS-Like**

Use the following moves: $(+1, -1, +2, -2, +3, -3, ...)$

If the treasure is in location $t$, the cost is:

$$ON = 4(1 + 2 + 3 + ... + t - 1) + t$$

and for location $(-t)$:

$$ON = 4(1 + 2 + 3 + ... + t - 1) + 3t$$

Thus, $ON = O(t^2)$ and $OPT = O(t) \Rightarrow t = \infty$

**Solution 2: Geometric Walk**

The moves: $(+1, -1, +2, -2, +4, -4, ..., +2^i, -2^i, ...)$

The worst location is in $-(2^k + \epsilon)$ for which the cost is:

$$ON = 4(1 + 2 + ... + 2^k) + 2 \cdot 2^{k+1} + 2^k + \epsilon$$

(since we stop when we find the treasure)

Sum of this geometric series: $4 \cdot 2^{k+1} - 4 + 5 \cdot 2^k + \epsilon = 13 \cdot 2^k + \epsilon$

$OPT = 2^k + \epsilon \Rightarrow$ Competitive ratio is 13.

**Solution 3,4,...**

$$(+1, -2, +4, -8, +16, ...)
(\pm q^0, -q^0, +q^1, -q^1, ...)
(\pm q^0, -q^1, +q^2, -q^3, ...)

And in general:

$$+x_0, -x_1, +x_2, -x_3, ...$$

$$0 < x_0 < x_2 < x_4 < ...$$

$$0 < x_1 < x_3 < x_5 < ...$$
If the treasure is in the positive direction, the cost:

\[ 2(x_0 + x_1 + x_2 + ... + x_{2k} + x_{2k+1}) + x_{2k} + \epsilon \]

letting \( n = 2k \)

\[ = 2(x_0 + ... + x_{n+1}) + x_n + \epsilon \]

and for the negative direction:

\[ 2(x_0 + x_1 + x_2 + ... + x_{2k+1} + x_{2k+2}) + x_{2k+1} + \epsilon \]

\[ = 2(x_0 + ... + x_{n+1}) + x_n + \epsilon \]

Thus, the same for both directions.
(Note that we assume that the treasure is after \( x_0 \))

**The competitive ratio:**

\[
\frac{ON}{OPT} = \frac{2(x_0 + ... + x_{n+1}) + x_n + \epsilon}{x_n + \epsilon}
\]

The nominator is larger, thus the supremum is obtained where \( \epsilon \) is minimal.
For \( s_n = x_0 + ... + x_n \):

\[
\sup_{\epsilon} \frac{2s_{n+1} + x_n}{x_n} = \frac{2s_{n+1}}{x_n} + 1
\]

Given a strategy, the competitive ratio is:

\[ a = \sup_n \frac{s_{n+1}}{x_n} \]

for \( x_i = q^i \): 

\[ a = \sup_n \frac{q^{n+2} - 1}{(q - 1)q^n} = \sup_n \frac{q^2}{q - 1} - \frac{1}{(q - 1)q^n} = \frac{q^2}{q - 1} \]

Where in the last step the quotient with \( n \) drops since it is negligible for a large \( n \).

Now we derive in order to find the minimum:

\[ t = \frac{q^2}{q - 1} \]

\[ t' = \frac{2q(q - 1) - q^2}{(q - 1)^2} = 0 \]

\[ q = 2 \]

\[ a = \frac{2^2}{2 - 1} = 4 \]

Hence the competitive ratio is \( 2a + 1 = 9 \).
**Theorem 3.3.** For any positive, monotonic (for simplicity) series, $x_i$:

\[ \forall \epsilon > 0 : \sup x_n \geq 4 - \epsilon \]

**Proof.** We assume that we get a ratio $a$, with $a > 4$.

Define $y_n = s_n/s_{n-1}$ and we get:

\[ a \geq \frac{s_{n+1}}{x_n} = \frac{s_{n+1}}{s_n - s_{n-1}} = \frac{s_{n+1}/s_n}{1 - s_{n-1}/s_n} = \frac{y_{n+1}}{1 - 1/y_n} \]

**Lemma:** $\forall \beta > 0$: $1 - 1/\beta \leq \beta/4$.

The proof is trivial: $4\beta - 4 \leq \beta^2 \Rightarrow 0 \leq (\beta - 2)^2$.

Using this lemma we get

\[ y_{n+1} \leq a(1 - 1/y_n) \leq \frac{a}{4} y_n \]

We know that $\frac{a}{4} \leq 1 - \epsilon$, so for $n \to \infty$ the sandwich theorem implies that $y_n \to 0$.

Hence $y_n < 1$ starting at some $n$.

But $y_n = s_n/s_{n-1} > 1$ and we get a contradiction.

Hence, we cannot obtain a competitive ratio under 4.  \qed
Scheduling

We are seeking to minimize the maximum load \( l_i = \sum_{j|A(i)=j} w_i \) over all machines.

A greedy algorithm: “Always assign a task to the least loaded machine.”

**Theorem 3.4.** The greedy algorithm for assigning tasks to identical machines is 2-competitive.

**Proof.** We only provide intuitive illustrated proof.

The following drawing of “horizon” represents the current load (y axis) for each machine (x axis).

The colored task is the task assigned most recently.

![Load Horizon Line](image)

Figure 1: Load Horizon Line

Basic observation tells us that \( OPT \geq w \) and \( OPT \geq h \) and thus

\[
ON = h + w \leq OPT + OPT = 2
\]