CS **Summer**

Online Algorithms

Authors: Shaleen Baral

Contents

1. Introduction

1.1. Approximation Algorithms

Definition 1.1.1 (Optimization Problem): An *optimization problem* Π is a 5-tuple $(\mathcal{I}, \mathcal{O}, s, q, g)$:

a. $\mathcal{I}$: the set of *instances.*
b. $\mathcal{O}$: the set of *solutions.*
c. $s : \mathcal{I} \to \mathcal{P}(\mathcal{O})$: for every instance $I \in \mathcal{I}$, $s(I) \subseteq \mathcal{O}$ denotes the set of feasible solutions for I.
d. $q : \mathcal{I} \times \mathcal{O} \to \mathbb{R}$: for every instance $I \in \mathcal{I}$ and every feasible solution $O \in s(I)$, $q(I, O)$ denotes the measure of I and O.
e. $g \in \{\max, \min\}$.

An *optimal solution* for an instance $I \in \mathcal{I}$ of Π is a solution $\mathtt{OPT}(I) \in s(I)$ such that

$$q(I, \mathtt{OPT}(I)) = g\{q(I, O) \mid O \in s(I)\}.$$

If $g = \min$, we call Π a *minimization problem* and refer to q as the *cost.* If $g = \max$ we call Π a *maximization problem* and refer to q as the *gain.*

Definition 1.1.2 (Consistent): An algorithm is *consistent* for an (optimization) problem Π if it computes a feasible solution for every given instance.

Definition 1.1.3 (Approximation Algorithms): Let Π be an optimization problem, and let $\mathtt{ALG}$ be a consistent algorithm for Π. For $r \geq 1$, $\mathtt{ALG}$ is an *r-approximation algorithm* for Π if, for every $I \in \mathcal{I}$

$$q(\mathtt{OPT}(I)) \leq r \cdot q(\mathtt{ALG}(I))$$

if Π is a maximization problem, or

$$q(\mathtt{ALG}(I)) \leq r \cdot q(\mathtt{ALG}(I))$$

if Π is a minimization problem.

The *approximation ratio* of $\mathtt{ALG}$ is defined as

$$r_{\mathtt{ALG}} = \inf\{r \geq 1 \mid \mathtt{ALG} \text{ is an } r\text{-approximation algorithm for } \Pi\}.$$

Remark: Sometimes we define r-approximation so that $r \in (0, 1]$.

Definition 1.1.4 (Simple Knapsack Problem): The *simple knapsack problem* is a maximization problem specified as follows:

An instance I is given by a sequence of $n + 1$ numbers– $w_1, w_2, ..., w_n, W$. A feasible set is any set $O \subseteq [n]$ such that

$$\sum_{i \in O} w_i \leq W.$$

The gain of a solution O and a corresponding instance I is given by

$$q(I, O) = \sum_{i \in O} w_i.$$

The goal is to maximize this gain.

Remark: Think of the numbers w_i as being weights of items and W as being the maximum weight capacity of the knapsack.

Algorithm 1: `KN-GREEDY`

```
$O = \emptyset$
$s = 0$
**sort** such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
**while** $i < n$ and $s + w_{i+1} \leq W$ **do**
    $O = O \cup w_{i+1}$
    $s = s + w_{i+1}$
    $i = i + 1$
**end**
**return** $O$
```

Proposition 1.1.1: `KN-Greedy` is a polynomial-time 2-approximation algorithm for the simple knapsack problem.

Proof: Running time is $\mathcal{O}(n)$, which is polynomial. The loop invariants ensure that the algorithm is consistent.

Now, we show that this is a 2-approximation algorithm. Consider the labeling of the weights after sorting i.e. assume $w_1 \geq w_2 \geq \cdots \geq w_n$. If the algorithm accepts all the items, then this is clearly the optimal solution too, $\texttt{OPT} = \texttt{ALG}$.

Suppose that the algorithm rejects the $(r+1)$-th item. If $\sum_{i=1}^{r} w_i \geq \frac{W}{2}$ then we are done as

$$\texttt{ALG} = \sum_{i=1}^{r} w_i \geq \frac{W}{2} \geq \frac{\texttt{OPT}}{2}.$$

Next, if $\sum_{i=1}^{r} w_i < \frac{W}{2}$ then note that $w_{r+1} \leq w_r < \frac{W}{2}$. Since the $(r+1)$-th item is rejected,

$$\texttt{ALG} = \sum_{i=1}^{r} w_i > W - w_{r+1} > \frac{W}{2} \geq \frac{\texttt{OPT}}{2}.$$

To see that this bound is tight with $r = 2$, we construct a family of instances whose approximation ratio approaches 2. Consider the instance with $n = 3$, with weights

$$\frac{W}{2} + 1, \frac{W}{2}, \frac{W}{2}$$

and maximum weight capacity W. Then, the algorithm always accepts the item with weight $\frac{W}{2} + 1$ and cannot accept any more whereas the optimum solution would accept the two items with weight $\frac{W}{2}$. The approximation ratio is then,

$$\frac{\texttt{OPT}}{\texttt{ALG}} = \frac{W}{\frac{W}{2} + 1} = 2 \cdot \frac{W}{W + 1}$$

As $W \to \infty$, $\texttt{OPT}/\texttt{ALG} \longrightarrow 2$.

□

Definition 1.1.5 (FPTAS): A *fully polynomial-time approximation scheme* is an algorithm that takes as input an $\varepsilon > 0$ and an instance of a (optimization) problem and returns an output value that is at least $(1 - \varepsilon)$ OPT and at most $(1 + \varepsilon)$ OPT. Importantly, the running time of the algorithm must be polynomial in both the size of the input and $\frac{1}{\varepsilon}$.

1.2. Online Algorithms

Definition 1.2.1 (Online Problem): An *online problem* Π is a 5-tuple $(\mathcal{I}, \mathcal{O}, s, q, g)$:

a. $\mathcal{I}$: the set of *instances*. Every instance $I \in \mathcal{I}$ is a sequence of *requests* $I = (x_1, x_2, ..., x_n)$ with $n \in \mathbb{N}$.
b. $\mathcal{O}$: the set of *solutions*. Every output $O \in \mathcal{O}$ is a sequence of *answers* $O = (y_1, y_2, ..., y_n)$ with $n \in \mathbb{N}$.
c. $s : \mathcal{I} \to \mathcal{P}(\mathcal{O})$: for every instance $I \in \mathcal{I}$, $s(I) \subseteq \mathcal{O}$ denotes the set of *feasible solutions* for I.
d. $q : \mathcal{I} \times \mathcal{O} \to \mathbb{R}$: for every instance $I \in \mathcal{I}$ and every feasible solution $O \in s(I)$, $q(I, O)$ denotes the measure of I and O.
e. $g \in \{\max, \min\}$.

An *optimal solution* for an instance $I \in \mathcal{I}$ of Π is a solution $\texttt{OPT}(I) \in s(I)$ such that

$$q(I, \texttt{OPT}(I)) = g\{q(I, O) \mid O \in s(I)\}.$$

If $g = \min$, we call Π a *online minimization problem* and refer to q as the *cost*. If $g = \max$ we call Π a *online maximization problem* and refer to q as the *gain*.

Definition 1.2.2 (Online Algorithm): Let Π be an online problem and let $I = (x_1, x_2, ..., x_n)$ be an instance of Π. An *online algorithm* ALG for Π computes the output $\texttt{ALG}(I) = (y_1, y_2, ..., y_n)$, where y_i only depends on $x_1, x_2, ..., x_i$ and $y_1, y_2, ..., y_{i-1}$; we also require $\texttt{ALG}(I) \in s(I)$, that is, $\texttt{ALG}(I)$ is a fesible solution for I.

Definition 1.2.3 (Competitive Ratio): Let Π be an online problem, and let ALG be a consistent online algorithm for Π. For $c \geq 1$, ALG is *c-competitive* for Π if there is a constant $\alpha \geq 0$ such that, for every instance $I \in \mathcal{I}$.

$$q(\texttt{OPT}(I)) \leq c \cdot q(\texttt{ALG}(I)) + \alpha$$

if Π is an online maximization problem, or

$$q(\texttt{ALG}(I)) \leq c \cdot q(\texttt{OPT}(I)) + \alpha$$

if Π is an online mimization problem. If these inequalities holds with $\alpha = 0$ we call ALG *strictly c-competitive*. ALG is *optimal* if it is strictly 1-competitive.

The *competitive ratio* is defined as

$$c_{\texttt{ALG}} = \inf\{c \geq 1 \mid \texttt{ALG} \text{ is } c\text{-competitive for } \Pi\}.$$

If the competitive ratio of ALG is constant and the best that is achievable by any online algorithm for Π, we call ALG *strongly c-competitive*.

Remark: Commonly, we refer to $\texttt{OPT}(I)$ as the *optimal offline solution*. It may not even be achievable online as it may require knowledge of the complete instance!

Remark: If the competitive ratio of an online algorithm ALG is at most c, where c is a constant, we call ALG *competitive* or if $\alpha = 0$, *strictly competitive*. If the algorithm does not possess a competitive ratio with an upper bound that is idenpednent of the input length, we call it *not competitive*. It is fine to call an online algorithm competitive if its competitive ratio depends on some parameter of the problem being studied. This classification isn't always clear cut but is still often helpful.

Why do we have the additive constant α when defining the competitive ratio? The additive constant allows us to ignore finitely many exceptional instances on which the online algorithm may perform poorly. Particularly, to prove lower bounds, the constant α forces us to construct infinitely many instances with increasing costs or gains. The following lemmas and propositions make this notion more explicit.

Lemma 1.2.1: Let Π be an online minimization problem and let $\mathcal{I} = \{I_1, I_2, ...\}$ be an infinite set of instances of Π such that $|I_i| \leq |I_{i+1}|$ and such that the number of different input lengths in $\mathcal{I}$ is infinite. Suppose further that $q(\texttt{OPT}) > 0$ on $\mathcal{I}$. If there is an increasing, unbounded function $c : \mathbb{N}^+ \to \mathbb{R}^+$ such that,

$$\frac{q(\texttt{ALG}(I_i))}{q(\texttt{OPT}(I_i))} \geq c(n) \text{ where } n = |I_i|,$$

then `ALG` isn't competitive.

Proof: Suppose for contradiction that `ALG` is competitive, that is

$$q(\texttt{ALG}(I_i)) \leq c' \cdot q(\texttt{OPT}(I_i)) + \alpha.$$

Then,

$$[c(n) - c'] \cdot q(\texttt{OPT}(I_i)) \leq \alpha.$$

This clearly contradicts the fact that α is a constant.

$\square$

Lemma 1.2.2: Let Π be an online minimization problem and let $\mathcal{I} = \{I_1, I_2, ...\}$ be an infinite set of instances of Π such that $|I_i| \leq |I_{i+1}|$ and such that the number of different input lengths in $\mathcal{I}$ is infinite. Suppose further that $q(\texttt{ALG}) > 0$ on $\mathcal{I}$. If there is an increasing, unbounded function $c : \mathbb{N}^+ \to \mathbb{R}^+$ such that,

$$\frac{q(\texttt{OPT}(I_i))}{q(\texttt{ALG}(I_i))} \geq c(n) \text{ where } n = |I_i|,$$

then `ALG` isn't competitive.

Remark: Since this is a minimization problem, $q(\texttt{OPT}) > 0$ also implies $q(\texttt{ALG}) > 0$.

Proof: Same idea as the prior proof.

$\square$

Proposition 1.2.1: Let Π be an online minimization problem, and let $\mathcal{I} = \{I_1, I_2, ...\}$ be an infinite set of instances of Π such that $|I_i| \leq |I_{i+1}|$, and such that the number of different input lengths in $\mathcal{I}$ is infinite. Let `ALG` be an online algorithm for Π. If there is some constant $c \geq 1$ such that

a. $\dfrac{q(\texttt{ALG}(I_i))}{q(\texttt{OPT}(I_i))} \geq c,$ for every $i \in \mathbb{N}^+$
b. $\lim_{i \to \infty} q(\texttt{OPT}(I_i)) = \infty$

then `ALG` isn't a $(c - \varepsilon)$-competitive online algorithm for Π, for any $\varepsilon > 0$.

Proof: Suppose for contradiction that ALG is a $(c - \varepsilon)$-competitive algorithm for some $\varepsilon > 0$. Then, there is a constant α such that

$$q(\texttt{ALG}(I_i)) \leq (c - \varepsilon) \cdot q(\texttt{OPT}(I_i)) + \alpha$$
$$\Longrightarrow \frac{q(\texttt{ALG}(I_i))}{q(\texttt{OPT}(I_i))} - \frac{\alpha}{q(\texttt{OPT}(I_i))} \leq c - \varepsilon$$

for every $i \in \mathbb{N}^+$. By a), the first term above is at least c and by b), we can find instances for which the second term is less than ε. Thus, we have a contradiction.

□

Proposition 1.2.2: Let Π be an online maximization problem, and let $\mathcal{I} = \{I_1, I_2, ...\}$ be an infinite set of instances of Π such that $|I_i| \leq |I_{i+1}|$, and such that the number of different input lengths in $\mathcal{I}$ is infinite. Let ALG be an online algorithm for Π. If there is some constant $c \geq 1$ such that

a. $\frac{q(\texttt{OPT}(I_i))}{q(\texttt{ALG}(I_i))} \geq c$, for every $i \in \mathbb{N}^+$
b. $\lim_{i \longrightarrow \infty} q(\texttt{OPT}(I_i)) = \infty$

then ALG isn't a $(c - \varepsilon)$-competitive online algorithm for Π, for any $\varepsilon > 0$.

Proof: Suppose for contradiction that ALG is a $(c - \varepsilon)$-competitive online algorithm for some $\varepsilon > 0$. Then, there is some constant α such that

$$\frac{q(\texttt{OPT}(I_i))}{q(\texttt{ALG}(I_i))} - \frac{\alpha}{q(\texttt{ALG}(I_i))} \leq c - \varepsilon$$

for every $i \in \mathbb{N}^+$. Furthermore, by a) and b), if $q(\texttt{ALG}(I_i))$ were bounded by a constant, it wouldn't be competitive at all. Thus, it is fair to assume that $\lim_{i \longrightarrow \infty} q(\texttt{ALG}(I_i)) = \infty$. By a), first term above is at least c and we can find instances for which the second term is less than ε. Thus, we have a contradiction.

□

Remark: Propositions 1.2.1 and 1.2.2 need to be carefully interpreted. For example, they don't rule out the possibility for a $\left(c - \frac{1}{n}\right)$-competitive algorithm.

1.3. Paging

Definition 1.3.1 (Paging): The *paging problem* is an online minimization problem.

Suppose there are $m \in \mathbb{N}^+$ memory pages $p_1, p_2, ..., p_m$, which are stored in the main memory. An instance is a sequence $I = (x_1, x_2, ..., x_n)$ such that $x_i \in \{p_1, p_2, ..., p_m\}$, for all $i \in [n]$. This means that page x_i is requested in time step T_i.

An online algorithm ALG for paging maintains a *cache* memory of size k with $k < m$, represented by the tuple $B_i = \left(p_{j_1}, p_{j_2}, ..., p_{j_k}\right)$ for time step T_i. Initially, the cache is initialized as $B_0 = (p_1, p_2, ..., p_k)$, that is, with the first k pages. If in some time step T_i, a page x_i is requested and $x_i \in B_{i-1}$, ALG outputs $y_i = 0$. Conversely, if $x_i \notin B_{i-1}$, ALG has to choose a page $p_j \in B_{i-1}$, which is then removed from the cache to make room for x_i. In this case, ALG outputs $y_i = p_j$ and the new cache content is $B_i = \left(B_{i-1} \setminus p_j\right) \cup x_i$.

The cost is defined as

$$q(\texttt{ALG}(I)) = |\{i \in [n] \mid y_i \neq 0\}|$$

and the goal is to minimize it.

Definition 1.3.2 (k-Phase Partition): Let $I = (x_1, x_2, ..., x_n)$ be an arbitrary instance of paging. A *k-phase partition of I* assigns the requests from I to consecutive disjoint phases $P_1, P_2, ..., P_N$ such that

a. Phase P_1 starts with the first request for a page that is not initially in the cache. Then, P_1 contains a maximum-length subsequence of I that contains at most k distinct pages.
b. For any i with $2 \leq i \leq N$, phase P_i is a maximum-length subsequence of I that starts right after P_{i-1} and again contains at most k distinct pages.

Algorithm 2: `FIFO`

The cache is organized as a queue. Whenever a page must be evicted, the one residing in the cache for the longest time is chosen. The first k pages may be removed arbirarily.

Proposition 1.3.1: `FIFO` is strictly k-competitive for paging.

Proof: Let $I = (x_1, x_2, ..., x_N)$ be any instance of paging and consider I's k-phase partition $P_1, P_2, ..., P_N$. WLOG, assume $x_1 \notin \{p_1, p_2, ..., p_k\}$.

□