CS 341: Algorithms

Lecture 6: Greedy algorithms, continued

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Minimizing lateness

The problem

Input:

- jobs $J_1, \ldots, J_n$ with processing times $t(1), \ldots, t(n)$ and deadlines $d(1), \ldots, d(n)$
- can only do one thing at a time

The problem

Input:

- jobs $J_1, \ldots, J_n$ with processing times $t(1), \ldots, t(n)$ and deadlines $d(1), \ldots, d(n)$
- can only do one thing at a time

Output:

- a **scheduling** of the jobs which **minimizes maximal lateness**
  - job J_i starts at time $s(i)$ (TBD) and finishes at $f(i) = s(i) + t(i)$
  - if $f(i) \geq d(i)$, lateness $\ell(i) = f(i) - d(i)$, otherwise $\mathbf{0}$
- maximal lateness $= \max_i \ell(i)$

Example: 3 jobs

- **prepare my slides:** need $\boldsymbol{t(1) = 4}$ hours, deadline $\boldsymbol{d(1) = 2}$ hours
- **write solutions to assignments:** need $\boldsymbol{t(2) = 6}$ hours, deadline $\boldsymbol{d(2) = 1}$ hour
- **bake a panettone:** need $\boldsymbol{t(3) = 10}$ hours, deadline $\boldsymbol{d(3) = 24}$ hours

Example: 3 jobs

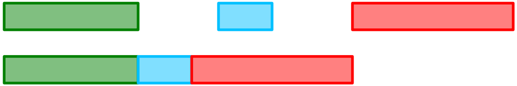
- **prepare my slides:** need $t(1) = 4$ hours, deadline $d(1) = 2$ hours
- **write solutions to assignments:** need $t(2) = 6$ hours, deadline $d(2) = 1$ hour
- **bake a panettone:** need $t(3) = 10$ hours, deadline $d(3) = 24$ hours

- **1, then 2, then 3:** latenesses $[2, 9, 0]$
- **2, then 1, then 3:** latenesses $[8, 5, 0]$ (optimal)

No breaks

Observation:

- if a scheduling has **idle time**, we can improve it by removing the breaks



- so the optimal has no idle time, and is given by a **permutation** of $[1, \ldots, n]$

A few attempts

Attempt 1:

- do short jobs first

A few attempts

Attempt 1:

- do short jobs first
- **no**, last example

A few attempts

Attempt 1:

- do short jobs first
- **no**, last example

Attempt 2:

- do jobs with little slack first — slack $= d(i) - t(i)$

A few attempts

Attempt 1:

- do short jobs first
- **no**, last example

Attempt 2:

- do jobs with little slack first $\qquad$ slack $= d(i) - t(i)$
- **no**

  take $\boldsymbol{t(1) = 8}$, $\boldsymbol{d(1) = 10}$ so $\boldsymbol{s(1) = 2}$ and $\boldsymbol{t(1) = 2}$, $\boldsymbol{d(2) = 5}$ so $\boldsymbol{s(2) = 3}$

A few attempts

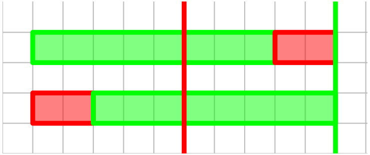
Attempt 1:

- do short jobs first
- **no**, last example

Attempt 2:

- do jobs with little slack first — slack $= d(i) - t(i)$
- **no**

  take $\boldsymbol{t(1) = 8}$, $\boldsymbol{d(1) = 10}$ so $\boldsymbol{s(1) = 2}$ and $\boldsymbol{t(1) = 2}$, $\boldsymbol{d(2) = 5}$ so $\boldsymbol{s(2) = 3}$



A few attempts

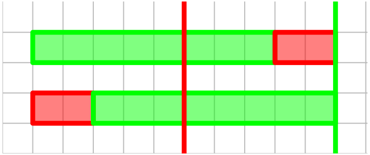
Attempt 1:

- do short jobs first
- **no**, last example

Attempt 2:

- do jobs with little slack first $\qquad$ slack $= d(i) - t(i)$
- **no**

  take $\boldsymbol{t(1) = 8}$, $\boldsymbol{d(1) = 10}$ so $\boldsymbol{s(1) = 2}$ and $\boldsymbol{t(1) = 2}$, $\boldsymbol{d(2) = 5}$ so $\boldsymbol{s(2) = 3}$



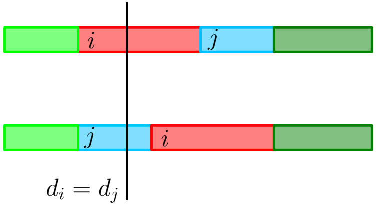
Attempt 3:

- do jobs in non-decreasing deadline order

Non-uniqueness

Observation:

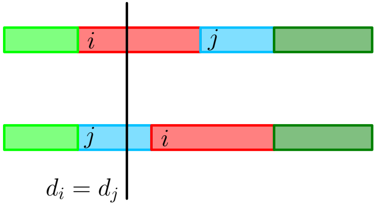
- if $d(i) = d(j)$, the orderings $[\ldots, \boldsymbol{i}, \boldsymbol{j}, \ldots]$ and $[\ldots, \boldsymbol{j}, \boldsymbol{i}, \ldots]$ have the same max-lateness (because the second job is the latest)



Non-uniqueness

Observation:

- if $d(i) = d(j)$, the orderings $[\ldots, \boldsymbol{i}, \boldsymbol{j}, \ldots]$ and $[\ldots, \boldsymbol{j}, \boldsymbol{i}, \ldots]$ have the same max-lateness (because the second job is the latest)

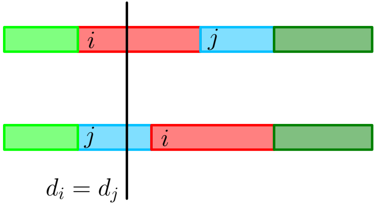


- so **all** orderings in non-decreasing deadline order have the same max-lateness

Non-uniqueness

Observation:

- if $d(i) = d(j)$, the orderings $[\ldots, \boldsymbol{i}, \boldsymbol{j}, \ldots]$ and $[\ldots, \boldsymbol{j}, \boldsymbol{i}, \ldots]$ have the same max-lateness (because the second job is the latest)



- so **all** orderings in non-decreasing deadline order have the same max-lateness

Definition:

- take a permutation $L = [e_1, \ldots, e_n]$ of $[1, \ldots, n]$
- **inversion**: a pair (i, j) with $i < j$ and $d(e_i) > d(e_j)$ (= an inversion in $[d(e_1), \ldots, d(e_n)]$ in the sense of lecture 3)
- no inversion $\iff$ L in non-decreasing deadline order

Correctness: exchange argument

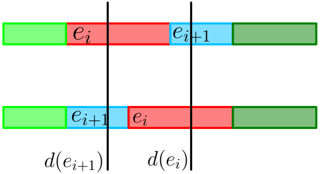
- let $L = [e_1, \ldots, e_n]$ be **any** permutation of $[1, \ldots, n]$
- suppose that L is **not** in non-decreasing order of deadlines
- want: $\mathbf{max\_lateness}(\boldsymbol{L}) \geq \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$

Correctness: exchange argument

- let $L = [e_1, \ldots, e_n]$ be **any** permutation of $[1, \ldots, n]$
- suppose that L is **not** in non-decreasing order of deadlines
- want: $\mathbf{max\_lateness}(\boldsymbol{L}) \geq \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$
- there exists i such that $\boldsymbol{d(e_i) > d(e_{i+1})}$.
- now, **swap $\boldsymbol{e_i}$ and $\boldsymbol{e_{i+1}}$ to get a permutation $\boldsymbol{L'}$.** What about $\mathrm{max\_lateness}(L')$?

Correctness: exchange argument

- let $L = [e_1, \ldots, e_n]$ be **any** permutation of $[1, \ldots, n]$
- suppose that L is **not** in non-decreasing order of deadlines
- want: $\mathbf{max\_lateness}(\boldsymbol{L}) \geq \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$
- there exists i such that $\boldsymbol{d(e_i) > d(e_{i+1})}$.
- now, **swap e_i and e_{i+1} to get a permutation L'.** What about max_lateness(L')?
- the lateness of e_{i+1} cannot increase (because we do e_{i+1} earlier than before), so at most max_lateness(L)
- the **new** lateness of e_i is **at most** the **old** lateness of e_{i+1}, so at most max_lateness(L)



Correctness: exchange argument

- let $L = [e_1, \ldots, e_n]$ be **any** permutation of $[1, \ldots, n]$
- suppose that L is **not** in non-decreasing order of deadlines
- want: $\mathbf{max\_lateness}(\boldsymbol{L}) \geq \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$
- there exists i such that $\boldsymbol{d(e_i) > d(e_{i+1})}$.
- now, **swap $\boldsymbol{e_i}$ and $\boldsymbol{e_{i+1}}$ to get a permutation $\boldsymbol{L'}$.** What about max_lateness(L')?
- the lateness of e_{i+1} cannot increase (because we do e_{i+1} earlier than before), so at most max_lateness(L)
- the **new** lateness of e_i is **at most** the **old** lateness of e_{i+1}, so at most max_lateness(L)
- nothing else changes, so max_lateness$(L') \leq$ max_lateness(L)

Correctness: exchange argument

- let $L = [e_1, \dots, e_n]$ be **any** permutation of $[1, \dots, n]$
- suppose that L is **not** in non-decreasing order of deadlines
- want: $\mathbf{max\_lateness}(\boldsymbol{L}) \geq \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$
- there exists i such that $\boldsymbol{d(e_i) > d(e_{i+1})}$.
- now, **swap e_i and e_{i+1} to get a permutation L'.** What about max_lateness(L')?
- the lateness of e_{i+1} cannot increase (because we do e_{i+1} earlier than before), so at most max_lateness(L)
- the **new** lateness of e_i is **at most** the **old** lateness of e_{i+1}, so at most max_lateness(L)
- nothing else changes, so max_lateness(L') $\leq$ max_lateness(L)
- we removed an inversion
- keep going: after at most $n(n-1)/2$ iterations, we have L_{ord} with **no inversion** and such that $\mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{ord}}) \leq \mathbf{max\_lateness}(\boldsymbol{L})$
- we saw that $\mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{ord}}) = \mathbf{max\_lateness}(\boldsymbol{L}_{\mathbf{greedy}})$

Fractional knapsack

The problem

Input:

- items $I_1, \ldots, I_n$ with weights $\boldsymbol{w_1, \ldots, w_n}$ and positive values $\boldsymbol{v_1, \ldots, v_n}$
- a capacity W

Output:

- **fractions** $E = e_1, \ldots, e_n$ such that
  - $0 \leq e_j \leq 1$ for all j
  - $e_1 w_1 + \cdots + e_n w_n \leq W$
  - $e_1 v_1 + \cdots + e_n v_n$ maximal

Example:

- $\boldsymbol{w_1 = 10, v_1 = 60}$, $\boldsymbol{w_2 = 30, v_2 = 90}$, $\boldsymbol{w_3 = 20, v_3 = 100}$
- $\boldsymbol{W = 50}$

The problem

Input:

- items $I_1, \ldots, I_n$ with weights $\boldsymbol{w_1, \ldots, w_n}$ and positive values $\boldsymbol{v_1, \ldots, v_n}$
- a capacity W

Output:

- **fractions** $E = e_1, \ldots, e_n$ such that
  - $0 \leq e_j \leq 1$ for all j
  - $e_1 w_1 + \cdots + e_n w_n \leq W$
  - $e_1 v_1 + \cdots + e_n v_n$ maximal

Example:

- $\boldsymbol{w_1 = 10, v_1 = 60}$, $\boldsymbol{w_2 = 30, v_2 = 90}$, $\boldsymbol{w_3 = 20, v_3 = 100}$
- $\boldsymbol{W = 50}$
- optimal is $\boldsymbol{e_1 = 1}$, $\boldsymbol{e_2 = 2/3}$, $\boldsymbol{e_3 = 1}$, total value **220**

The problem

Input:

- items $I_1, \ldots, I_n$ with weights $\boldsymbol{w_1, \ldots, w_n}$ and positive values $\boldsymbol{v_1, \ldots, v_n}$
- a capacity W

Output:

- **fractions** $E = e_1, \ldots, e_n$ such that
  - $0 \leq e_j \leq 1$ for all j
  - $e_1 w_1 + \cdots + e_n w_n \leq W$
  - $e_1 v_1 + \cdots + e_n v_n$ maximal

Remark:

- 0/1-**version:** $e_j \in \{0, 1\}$ for all j
- dynamic programming

The knapsack should be full

Remark:

- if $\sum_i w_i < W$, just take all $e_i = 1$
- so assume $\boldsymbol{\sum_i w_i \geq W}$

The knapsack should be full

Remark:

- if $\sum_i w_i < W$, just take all $e_i = 1$
- so assume $\sum_i w_i \geq W$

Observation:

- suppose we have an assignment with $\sum_i e_i w_i < W$
- then some e_i must be **less than** 1
- so we can increase the value by increasing this e_i

The knapsack should be full

Remark:

- if $\sum_i w_i < W$, just take all $e_i = 1$
- so assume $\boldsymbol{\sum_i w_i \geq W}$

Observation:

- suppose we have an assignment with $\boldsymbol{\sum_i e_i w_i < W}$
- then some e_i must be **less than** 1
- so we can increase the value by increasing this e_i

Consequence:

- it is enough to consider assignments with $\boldsymbol{\sum_i e_i w_i = W}$

A few attempts

Attempt 1:

- pack with items in **decreasing value** v_i

A few attempts

Attempt 1:

- pack with items in **decreasing value** v_i
- **no**, previous example (we get $[0, 1, 1]$ with total value **190**)

A few attempts

Attempt 1:

- pack with items in **decreasing value** v_i
- **no**, previous example (we get $[0, 1, 1]$ with total value **190**)

Attempt 2:

- pack with items in **increasing weight** w_i

A few attempts

Attempt 1:

- pack with items in **decreasing value** v_i
- **no**, previous example (we get $[0, 1, 1]$ with total value **190**)

Attempt 2:

- pack with items in **increasing weight** w_i
- **no**: $\boldsymbol{W = 10}$, $\boldsymbol{w_1 = 10, v_1 = 100}$, $\boldsymbol{w_2 = 5, v_2 = 1}$

A few attempts

Attempt 1:

- pack with items in **decreasing value** v_i
- **no**, previous example (we get $[0, 1, 1]$ with total value $\mathbf{190}$)

Attempt 2:

- pack with items in **increasing weight** w_i
- **no**: $\boldsymbol{W = 10}$, $\boldsymbol{w_1 = 10, v_1 = 100}$, $\boldsymbol{w_2 = 5, v_2 = 1}$

Attempt 3:

- pack with items in **non-increasing "value per kilo"** $\boldsymbol{v_i/w_i}$
- first example $\mathbf{[6, 3, 5]}$, second example $\mathbf{[10, 1/5]}$

Pseudo-code

GreedyKnapsack$(\boldsymbol{v}, \boldsymbol{w}, W)$

$E \leftarrow [0, \ldots, 0]$
sort items by non-increasing order of v_i/w_i
for $k = 1, \ldots, n$ **do**
    if $w_k < W$ **then**
        $E[k] \leftarrow 1$
        $W \leftarrow W - w_k$
    else
        $E[k] \leftarrow W/w_k$
        return

Remark: output is $S = [1, \ldots, 1, e_k, 0, \ldots, 0]$

Runtime: $O(n \log(n))$

Correctness: exchange argument

- let $E = [e_1, \ldots, e_n]$ be the **output**, with $\boldsymbol{\sum e_i w_i = W}$
- let $S = [s_1, \ldots, s_n]$ be **any assignment**, with $\boldsymbol{\sum s_i w_i = W}$
- assume that $\boldsymbol{S \neq E}$, want $\mathbf{value}(\boldsymbol{E}) \geq \mathbf{value}(\boldsymbol{S})$

Correctness: exchange argument

- let $E = [e_1, \ldots, e_n]$ be the **output**, with $\boldsymbol{\sum e_i w_i = W}$
- let $S = [s_1, \ldots, s_n]$ be **any assignment**, with $\boldsymbol{\sum s_i w_i = W}$
- assume that $\boldsymbol{S \neq E}$, want $\mathbf{value}(\boldsymbol{E}) \geq \mathbf{value}(\boldsymbol{S})$
- let i be the **first** index with $\boldsymbol{e_i \neq s_i}$
- greedy strategy: $\boldsymbol{e_i > s_i}$
- because their weights are the same, there is $\boldsymbol{j > i}$ with $\boldsymbol{s_j > e_j}$

Correctness: exchange argument

- let $E = [e_1, \ldots, e_n]$ be the **output**, with $\boldsymbol{\sum e_i w_i = W}$
- let $S = [s_1, \ldots, s_n]$ be **any assignment**, with $\boldsymbol{\sum s_i w_i = W}$
- assume that $\boldsymbol{S \neq E}$, want $\textbf{value}(\boldsymbol{E}) \geq \textbf{value}(\boldsymbol{S})$
- let i be the **first** index with $\boldsymbol{e_i \neq s_i}$
- greedy strategy: $\boldsymbol{e_i > s_i}$
- because their weights are the same, there is $\boldsymbol{j > i}$ with $\boldsymbol{s_j > e_j}$
- set $\boldsymbol{s'_i = s_i + \alpha / w_i}$ and $\boldsymbol{s'_j = s_j - \alpha / w_j}$, for α TBD > 0, all other $\boldsymbol{s'_k = s_k}$
- in any case, $\boldsymbol{\sum s'_i w_i = W}$ and $\textbf{value}(\boldsymbol{S'}) \geq \textbf{value}(\boldsymbol{S})$

Correctness: exchange argument

- let $E = [e_1, \dots, e_n]$ be the **output**, with $\sum e_i w_i = W$
- let $S = [s_1, \dots, s_n]$ be **any assignment**, with $\sum s_i w_i = W$
- assume that $S \neq E$, want $\textbf{value}(E) \geq \textbf{value}(S)$
- let i be the **first** index with $e_i \neq s_i$
- greedy strategy: $e_i > s_i$
- because their weights are the same, there is $j > i$ with $s_j > e_j$
- set $s'_i = s_i + \alpha / w_i$ and $s'_j = s_j - \alpha / w_j$, for α TBD > 0, all other $s'_k = s_k$
- in any case, $\sum s'_i w_i = W$ and $\textbf{value}(S') \geq \textbf{value}(S)$
- choose **the first** α such that **either** $s'_i = e_i$ **or** $s'_j = e_j$

$$\alpha = \min(w_i(e_i - s_i), w_j(s_j - e_j))$$

- we found S' with **one more common entry** with E, and $\textbf{value}(S') \geq \textbf{value}(S)$
- if $S' \neq E$, repeat, $\dots$, until $S''''^{\cdots} = E$