CSE2310 Algorithm Design

Lecture 2: Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2

TU Delft

You are here

The course so far

- Course organisation, refreshers, introduction to greedy algorithms

Today's content

- Second proof for "Selecting Breakpoint" (camping sites)
- New problems: Interval Scheduling & Scheduling to Minimize Lateness
- Running experiments

The future

- Greedy (week 2): MSTs & clustering, Huffman encoding
- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

The start of your journeys

By now we expect that you have:

- The required skills to start (or a plan to quickly recover them!) by having completed:
  - Module "Welcome to Algorithm Design"
  - Until "Lecture 1" of Module "Greedy"
- Completed until "Lecture 2" of Module "Greedy"

The start of your journeys

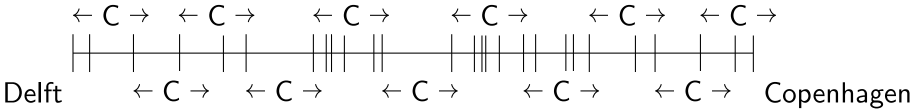
By now we expect that you have:

- The required skills to start (or a plan to quickly recover them!) by having completed:
  - Module "Welcome to Algorithm Design"
  - Until "Lecture 1" of Module "Greedy"
- Completed until "Lecture 2" of Module "Greedy"

- If you feel this took you too much time
  - check your prerequisite knowledge and skills from courses like R&L and ADS
  - work together
  - try out another path (don't do everything!)
  - ask for help: in lab sessions, or post them on Answers!

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

But is it optimal?

Proving techniques

1. Proof by induction: Greedy stays ahead
    - With equal number of stops, Greedy is at least as close to goal (Copenhagen) as optimal solution (Lemma)
    - Greedy thus is optimal (Theorem)
2. Or, proof by contradiction: reason about optimal solution most similar to greedy

TU Delft

But is it optimal?

Proving techniques

1. Proof by induction: Greedy stays ahead
   - With equal number of stops, Greedy is at least as close to goal (Copenhagen) as optimal solution (Lemma)
   - Greedy thus is optimal (Theorem)
2. Or, proof by contradiction: reason about optimal solution most similar to greedy

Notation

- Let $0 = g_1 < g_2 < \dots g_p = L$ denote the campsites chosen by greedy.
- Let $0 = f_1 < f_2 < \dots f_q = L$ denote the campsites in an optimal solution.

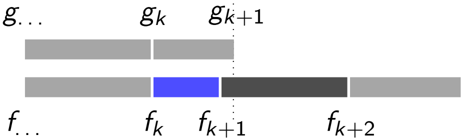
Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k+1$ it holds that $f_{k+1} \leq g_{k+1}$.
    - We know that $f_k \leq g_k$ by the IH.
    - Greedy selects g_{k+1} to be the campsite within reach as far from g_k as possible.
    - Campsite f_{k+1} cannot be further than the farthest reachable from f_k, and this is definitely within reach from g_k because of the IH. So $f_{k+1} \leq g_{k+1}$. □



But is it optimal?

Using the "greedy stays ahead" lemma

Theorem

The greedy algorithm is optimal.

Proof by contradiction.

- Let k be the number of campsites selected by greedy, and m the number of campsites in an optimal solution f.
- Suppose Greedy is not optimal. So $k > m$ and thus $g_m < f_m$.
- However, $f_m \leq g_m$ with the Lemma from the previous slide.[a]
- Contradiction with $g_m < f_m$. So Greedy must be optimal.

$\square$

[a] A Lemma is like a sub-routine.

Proof by contradiction: Exchange Argument

An alternative to the two-part proof before

Theorem

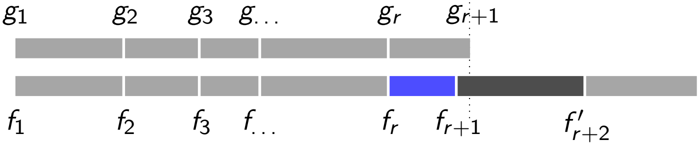
The greedy algorithm is optimal.

Proof by contradiction.

- Assume greedy is not optimal.
- Proof *idea*:
  - Reason about some specific optimal solution that is *as similar to the Greedy solution as possible*.
  - Show that an optimal solution exists that is even more similar.
- Contradiction! So greedy is optimal. □

Proof by contradiction: Exchange Argument

An alternative to the two-part proof before



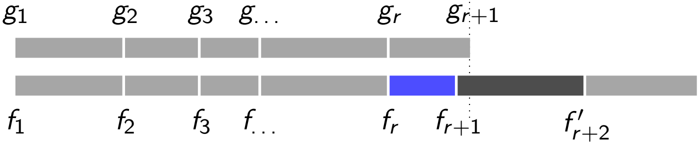
Proof by contradiction.

- Assume greedy is not optimal.
- Proof *idea*:
  - Reason about some specific optimal solution that is *as similar to the Greedy solution as possible*.
  - Show that an optimal solution exists that is even more similar.
- Contradiction! So greedy is optimal.

□

Proof by contradiction: Exchange Argument

An alternative to the two-part proof before



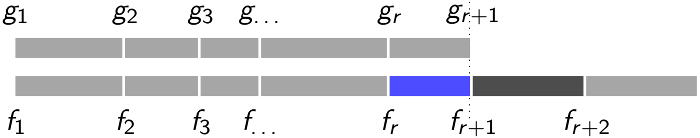
Proof by contradiction.

- Assume greedy is not optimal.
- Let $0 = f_1 < f_2 < \dots f_q = L$ denote the campsites in an optimal solution where $f_1 = g_1, f_2 = g_2, \dots f_r = g_r$ for the *largest possible value* of r. Meaning $f_{r+1} \neq g_{r+1}$.
- Note that $g_{r+1} \geq f_{r+1}$ by greedy choice of the algorithm, so $g_{r+1} > f_{r+1}$.

- Contradiction! So greedy is optimal.

Proof by contradiction: Exchange Argument

An alternative to the two-part proof before



Proof by contradiction.

- Assume greedy is not optimal.
- Let $0 = f_1 < f_2 < \dots f_q = L$ denote the campsites in an optimal solution where $f_1 = g_1, f_2 = g_2, \dots f_r = g_r$ for the *largest possible value* of r. Meaning $f_{r+1} \neq g_{r+1}$.
- Note that $g_{r+1} \geq f_{r+1}$ by greedy choice of the algorithm, so $g_{r+1} > f_{r+1}$.
- We can replace f_{r+1} by g_{r+1} (and maintain optimality), because
  - distance of $f'_{r+1} = g_{r+1}$ from f_r is same $g_{r+1} - g_r \leq C$
  - distance of $f'_{r+1} = g_{r+1}$ to f_{r+2} is less than $f_{r+2} - f_{r+1}$ (and thus also $\leq C$)
- Thus we have created an optimal solution same as greedy for the first $r + 1$ camp sites.
- Contradiction! So greedy is optimal. □

Interval Scheduling

Not quite the same as interval training

Image By: *Jamie Pitcher*

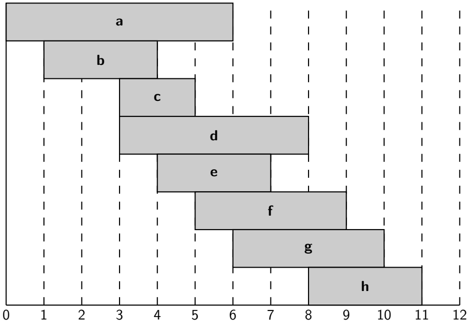
Interval scheduling

The problem

Problem: Interval scheduling (activity selection)

- Job (activity) j interval is given: starts at $s(j)$ and finishes at $f(j)$.
- Two jobs are compatible if they do not overlap.

What is the maximum number of compatible intervals (jobs)?



A greedy algorithm?

Let's just do it!

Please take your browser to `vevox.com` and use Session ID `191-417-320`

Greedy template

Consider the jobs in some order. Take each job provided it's compatible with the ones already taken.

Question: What order though?

- A. (Earliest start time) Consider jobs in ascending order of start time $s(i)$.
- B. (Earliest finish time) Consider jobs in ascending order of finish time $f(i)$.
- C. (Shortest interval) Consider jobs in ascending order of interval length $f(i) - s(i)$.
- D. (Fewest conflicts) For each job, count the number of conflicting jobs $c(i)$. Schedule in ascending order of conflicts $c(i)$.
- E. I don't know.

Three of those don't work

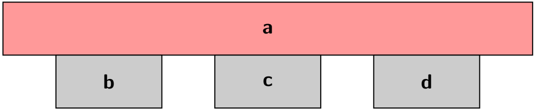
One counterexample each

Earliest start time?

Three of those don't work

One counterexample each

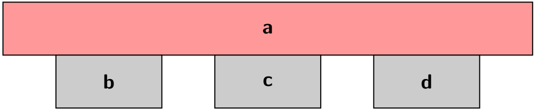
Earliest start time?



Three of those don't work

One counterexample each

Earliest start time?

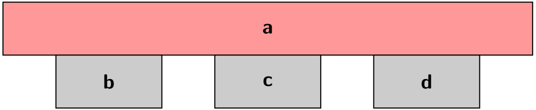


Shortest interval?

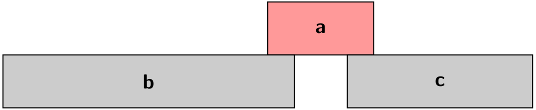
Three of those don't work

One counterexample each

Earliest start time?



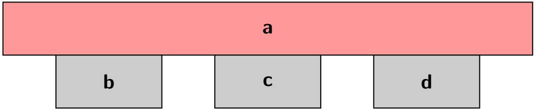
Shortest interval?



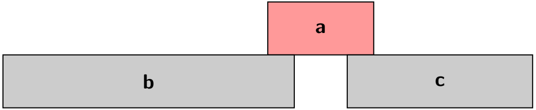
Three of those don't work

One counterexample each

Earliest start time?



Shortest interval?



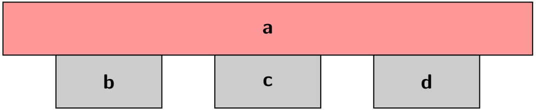
Fewest conflicts?

TU Delft

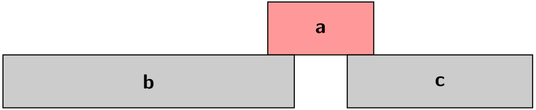
Three of those don't work

One counterexample each

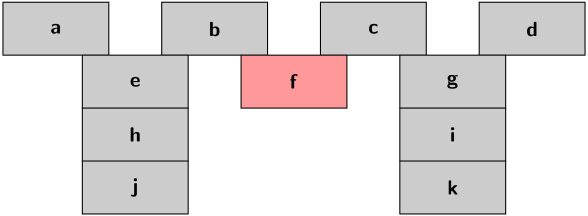
Earliest start time?



Shortest interval?



Fewest conflicts?



The algorithm at work

We'll refer to this algorithm as the "Greedy" algorithm

sort jobs by finish time so that
$f(1) \leq f(2) \leq \cdots \leq f(n)$
$A \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
  if job i is compatible with A **then**
    $A \leftarrow A \cup \{i\}$
return A

The algorithm at work

We'll refer to this algorithm as the "Greedy" algorithm

sort jobs by finish time so that
$f(1) \leq f(2) \leq \cdots \leq f(n)$
$A \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
    if job i is compatible with A **then**
        $A \leftarrow A \cup \{i\}$
return A

Question: Late, I'm late!

What is the tightest worst-case upper bound on the runtime?

The algorithm at work

We'll refer to this algorithm as the "Greedy" algorithm

sort jobs by finish time so that
$f(1) \leq f(2) \leq \cdots \leq f(n)$
$A \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
    if job i is compatible with A **then**
        $A \leftarrow A \cup \{i\}$
return A

Question: Late, I'm late!

What is the tightest worst-case upper bound on the runtime?

Answer: Implementation

Sorting (merge sort) takes $O(n \log n)$ time, the rest is linear:

- Remember job i^* that was added last to A.
- Job i is compatible with A if $s(i) \geq f(i^*)$.

Proving correctness and optimality

Proof outline

- Correctness: only compatible jobs are selected.
- Optimality:
  - Idea 1: Reason about optimal selection as much the same as possible to the greedy one. (Try this one at home!)
  - Idea 2: Greedy stays ahead

TU Delft

Proving correctness and optimality

Proof outline

- Correctness: only compatible jobs are selected.
- Optimality:
  - Idea 1: Reason about optimal selection as much the same as possible to the greedy one. (Try this one at home!)
  - Idea 2: Greedy stays ahead

Greedy stays ahead

- Show *with induction* that Greedy schedule with r jobs has no later finish time than any other schedule with r jobs (for every $r \leq k$, where k is total number of jobs in Greedy schedule).
- Use this to arrive at a contradiction.

Proof by induction: Greedy stays ahead

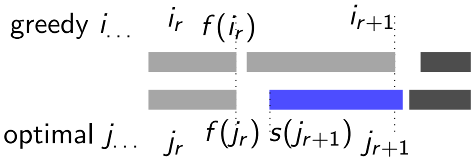
Based on page 120 of the book

Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

Please try this now yourself (5 minutes). $\square$



Proof by induction: Greedy stays ahead

Based on page 120 of the book

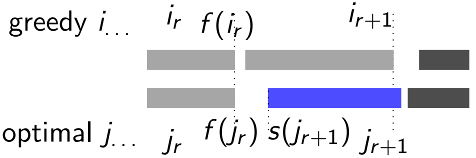
Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

- Base case ($r = 1$):
- Hypothesis:
- Induction step:

$\square$



Proof by induction: Greedy stays ahead

Based on page 120 of the book

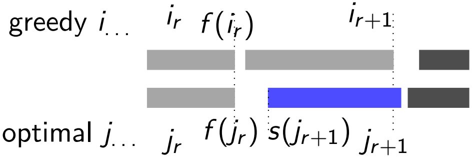
Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

- Base case ($r = 1$):
- Hypothesis: Suppose that for some r it holds that $f(i_r) \leq f(j_r)$.
- Induction step: To prove: for $r+1$ it holds that $f(i_{r+1}) \leq f(j_{r+1})$.

$\square$



Proof by induction: Greedy stays ahead

Based on page 120 of the book

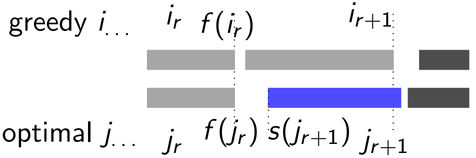
Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

- Base case ($r = 1$): Job i_1 is chosen and thanks to order of jobs, $f(i_1) \leq f(j_1)$.
- Hypothesis: Suppose that for some r it holds that $f(i_r) \leq f(j_r)$.
- Induction step: To prove: for $r+1$ it holds that $f(i_{r+1}) \leq f(j_{r+1})$.

$\square$



Proof by induction: Greedy stays ahead

Based on page 120 of the book

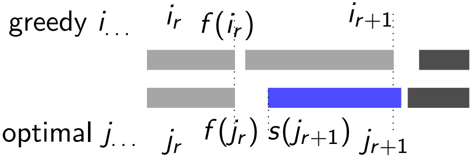
Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

- Base case ($r = 1$): Job i_1 is chosen and thanks to order of jobs, $f(i_1) \leq f(j_1)$.
- Hypothesis: Suppose that for some r it holds that $f(i_r) \leq f(j_r)$.
- Induction step: To prove: for $r + 1$ it holds that $f(i_{r+1}) \leq f(j_{r+1})$.
  We know that $f(j_r) \leq s(j_{r+1})$ as the jobs do not overlap.
  So $f(i_r) \leq s(j_{r+1})$ by the IH.
  So Greedy can take j_{r+1}, but since Greedy chooses smallest end time $f(i_{r+1}) \leq f(j_{r+1})$.

$\square$



Proof by induction: Greedy stays ahead

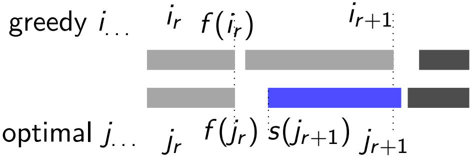
Based on page 120 of the book

Lemma 4.2

$\forall r \leq k : f(i_r) \leq f(j_r)$, with i denoting Greedy schedule and j some (optimal) one.

Proof by induction.

- Base case ($r = 1$): Job i_1 is chosen and thanks to order of jobs, $f(i_1) \leq f(j_1)$.
- Hypothesis: Suppose that for some r it holds that $f(i_r) \leq f(j_r)$.
- Induction step: To prove: for $r + 1$ it holds that $f(i_{r+1}) \leq f(j_{r+1})$.
  We know that $f(j_r) \leq s(j_{r+1})$ as the jobs do not overlap.
  So $f(i_r) \leq s(j_{r+1})$ by the IH.
  So Greedy can take j_{r+1}, but since Greedy chooses smallest end time $f(i_{r+1}) \leq f(j_{r+1})$.
- With induction thus $\forall r \leq k : f(i_r) \leq f(j_r)$. $\square$



Great, but what about the algorithm?

Theorem 4.3

Greedy algorithm is optimal (i.e. has the most jobs)

Proof by contradiction.

Let k be the number of jobs selected by Greedy in i, and m the number of jobs in some optimal schedule j. Suppose Greedy is not optimal, that is $k < m$.

Great, but what about the algorithm?

Theorem 4.3

Greedy algorithm is optimal (i.e. has the most jobs)

Proof by contradiction.

Let k be the number of jobs selected by Greedy in i, and m the number of jobs in some optimal schedule j. Suppose Greedy is not optimal, that is $k < m$.

However for all $r \leq k$ it holds that $f(i_r) \leq f(j_r)$ by Lemma 4.2.
In particular: $f(i_k) \leq f(j_k)$.
Thus there must be some job j_{k+1} in optimal solution j which starts after j_k, and thus after $f(i_k)$.

Great, but what about the algorithm?

Theorem 4.3

Greedy algorithm is optimal (i.e. has the most jobs)

Proof by contradiction.

Let k be the number of jobs selected by Greedy in i, and m the number of jobs in some optimal schedule j. Suppose Greedy is not optimal, that is $k < m$.

However for all $r \leq k$ it holds that $f(i_r) \leq f(j_r)$ by Lemma 4.2.
In particular: $f(i_k) \leq f(j_k)$.
Thus there must be some job j_{k+1} in optimal solution j which starts after j_k, and thus after $f(i_k)$.
But then after Greedy inserted i_k, there was another compatible job.
This contradicts Greedy having only k jobs. $\square$

Scheduling to minimize maximum lateness

Image From: *pxhere*

Scheduling to minimizing maximum lateness

You must do all jobs/assignments (for your chosen courses)

Problem: Minimizing lateness problem

- Single resource processes one job at a time.
- Job j requires t_j units of time and is due at d_j.
- If j starts at time s_j then it finishes at $f_j = s_j + t_j$.
- Lateness of j is $\mathscr{L}_j = \max(0, f_j - d_j)$. Maximum lateness: $\max_j(\mathscr{L}_j)$.

The goal is to *minimize* the maximum lateness, i.e.: minimize $\max_j(\mathscr{L}_j)$.

Scheduling to minimizing maximum lateness

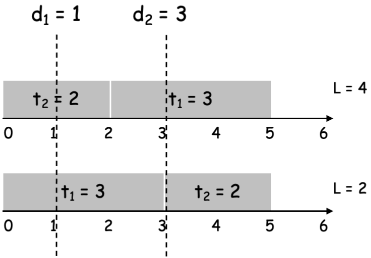
You must do all jobs/assignments (for your chosen courses)

Example: Two jobs, two orders

Take two jobs: $t_1 = 3, d_1 = 1$ and $t_2 = 2, d_2 = 3$.

Scheduling to minimizing maximum lateness

You must do all jobs/assignments (for your chosen courses)



Example: Two jobs, two orders

Take two jobs: $t_1 = 3, d_1 = 1$ and $t_2 = 2, d_2 = 3$.

- Job 2 first: $\mathscr{L}_1 = 4, \mathscr{L}_2 = 0$.
- Job 1 first: $\mathscr{L}_1 = 2, \mathscr{L}_2 = 2$.

Sorting orders again

Two jobs, two orders

Take two jobs: $t_1 = 3, d_1 = 1$ and $t_2 = 2, d_2 = 3$.

- Job 1 first: $\mathscr{L}_1 = 2, \mathscr{L}_2 = 2$.
- Job 2 first: $\mathscr{L}_1 = 4, \mathscr{L}_2 = 0$.

Question: What order?

A. (Shortest processing time first) Consider jobs in ascending order of processing time t_j (least work first).

B. (Earliest due date first) Consider jobs in ascending order of due date d_j (nearest due date).

C. (Smallest slack) Consider jobs in ascending order of slack $d_j - t_j$ (least time to start to make due date).

D. I don't know.

More counterexamples

Shortest processing time first

More counterexamples

Shortest processing time first

Take two jobs: $t_1 = 1, d_1 = 100$ and $t_2 = 10, d_2 = 10$.

- Job 1 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 1$.
- Job 2 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 0$.

More counterexamples

Shortest processing time first

Take two jobs: $t_1 = 1, d_1 = 100$ and $t_2 = 10, d_2 = 10$.

- Job 1 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 1$.
- Job 2 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 0$.

Smallest slack

More counterexamples

Shortest processing time first

Take two jobs: $t_1 = 1, d_1 = 100$ and $t_2 = 10, d_2 = 10$.

- Job 1 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 1$.
- Job 2 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 0$.

Smallest slack

Take two jobs: $t_1 = 1, d_1 = 2$ and $t_2 = 10, d_2 = 10$.

- Job 1 first: $\mathscr{L}_1 = 0, \mathscr{L}_2 = 1$.
- Job 2 first: $\mathscr{L}_1 = 9, \mathscr{L}_2 = 0$.

A greedy algorithm

Just like a student

sort n jobs by due date so that $d_1 \leq d_2 \leq \cdots \leq d_n$
$t \leftarrow 0$
for $j \leftarrow 1$ to n **do**
    $s_j \leftarrow t$
    $f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$
return intervals $\{[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]\}$

Earliest due date first

Observation: The greedy schedule has no idle time.

Observation: There is an optimal schedule with no idle time (as there is no penalty for removing it!)

Proof idea

Prove that earliest-due-date-first is optimal by an exchange argument:

- Take an optimal schedule that is as much as greedy as possible.
- Now make it even more similar to the greedy schedule without losing optimality.
- A contradiction!

Inversions

Definition (Inversions)

An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j is scheduled before i.

Observation: Greedy schedule has no inversions.

Inversions

Definition (Inversions)

An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j is scheduled before i.

Question: Swapping inversion

Suppose that two adjacent inverted jobs j and i with $d_i < d_j$ are swapped, what happens to the maximum lateness?

- A. The maximum lateness cannot become smaller.
- B. The maximum lateness cannot become larger.
- C. The maximum lateness always stays the same.
- D. I don't know.

Inversions

Definition (Inversions)

An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j is scheduled before i.

(Swapping inversion) Claim: Swapping two adjacent inverted jobs reduce the number of inversions by one and does not increase the maximum lateness.

Inversions

Definition (Inversions)

An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j is scheduled before i.

(Swapping inversion) Claim: Swapping two adjacent inverted jobs reduce the number of inversions by one and does not increase the maximum lateness.

Proof.

Let $\mathscr{L}$ be the lateness before the swap, and let $\mathscr{L}'$ be it after the swap.

- $\mathscr{L}'_k = \mathscr{L}_k$ for all $k \neq i, j$ (lateness for all others is the same).
- $\mathscr{L}'_i \leq \mathscr{L}_i$ as i is now done earlier.

Inversions

Definition (Inversions)

An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j is scheduled before i.

(Swapping inversion) Claim: Swapping two adjacent inverted jobs reduce the number of inversions by one and does not increase the maximum lateness.

Proof.

Let $\mathscr{L}$ be the lateness before the swap, and let $\mathscr{L}'$ be it after the swap.

- $\mathscr{L}'_k = \mathscr{L}_k$ for all $k \neq i, j$ (lateness for all others is the same).
- $\mathscr{L}'_i \leq \mathscr{L}_i$ as i is now done earlier.
- $$\begin{aligned} \mathscr{L}'_j &= f'_j - d_j && \text{Definition} \\ &= f_i - d_j && j \text{ finishes at time } f_i \text{ when swapped} \\ &\leq f_i - d_i && d_i < d_j \\ &\leq \mathscr{L}_i && \text{Definition} \end{aligned}$$

Inversions part 2

Claim: If a schedule (with no idle time) has an inversion, then it has one with a pair of inverted jobs scheduled consecutively (adjacent jobs).

Proof.

Inversions part 2

Claim: If a schedule (with no idle time) has an inversion, then it has one with a pair of inverted jobs scheduled consecutively (adjacent jobs).

Proof.

- Suppose there is an inversion.
- There is a pair of jobs i and j such that $d_i < d_j$, but j schedule before i.
- Walk through the schedule from j to i.
- Increasing due dates (= no inversions), at some point due date decreases.

$\square$

Putting it all together

Theorem

Greedy schedule S is optimal.

Proof.

Proof *idea:*

- Suppose S is not optimal.
- Take an optimal schedule S^* that is as much like greedy as possible.
- Change to look like greedy schedule (less inversions) without losing optimality.

lft

Putting it all together

Theorem

Greedy schedule S is optimal.

Proof.

- Suppose S is not optimal.
- Let S^* be an optimal schedule with the least inversions and no idle time.
- Thus $S \neq S^*$.

□

Putting it all together

Theorem

Greedy schedule S is optimal.

Proof.

- Suppose S is not optimal.
- Let S^* be an optimal schedule with the least inversions and no idle time.
- Thus $S \neq S^*$.
- If S^* has no inversions, then the lateness is the same as S. Contradiction!
- If S^* has an inversion,

□

Putting it all together

Theorem

Greedy schedule S is optimal.

Proof.

- Suppose S is not optimal.
- Let S^* be an optimal schedule with the least inversions and no idle time.
- Thus $S \neq S^*$.
- If S^* has no inversions, then the lateness is the same as S. Contradiction!
- If S^* has an inversion, let $i - j$ be an adjacent inversion (which exists!).
  - Swapping i and j does not increase the maximum lateness and decreases the number of inversions (see Swapping inversion Claim).
  - This contradicts the definition of S^*.
- Thus S is an optimal schedule.

□

Old exam question

5 minutes (+5 minutes)

Problem: Average end time

Give a greedy algorithm that for a series of n jobs with runtimes t_1 through t_n, creates a schedule for a machine such that the average finishing time is minimised. Determine also the end times $f_1, f_2, \ldots f_n$ in the algorithm.

Old exam question

5 minutes (+5 minutes)

Problem: Average end time

Give a greedy algorithm that for a series of n jobs with runtimes t_1 through t_n, creates a schedule for a machine such that the average finishing time is minimised. Determine also the end times $f_1, f_2, \ldots f_n$ in the algorithm.

Answer: Just sort

Sort the tasks in ascending order of t_i and renumber the indices.

$f_0 \leftarrow 0$

for $j \leftarrow 1$ to n **do**

$\quad f_j \leftarrow f_{j-1} + t_j$

Experiments!

https://www.youtube.com/watch?v=9kf51FpBuXQ

Image By: *Pixabay*

Experimental evaluation

Theory, theory, theory

- Worst-case analysis
- Best-case (harder!)
- Average-case (very hard!)

Experimental evaluation

Theory, theory, theory

- Worst-case analysis
- Best-case (harder!)
- Average-case (very hard!)

But all in the limit, for an arbitrary problem instance.

Question: Practice, practice, practice

1. What is the runtime on my practical problem instance?
2. For what problem size/properties is what algorithm really faster? (Relates to hidden constants in big-Oh/Θ)

Experimental studies

1. Write a program implementing the algorithm.
2. Run the program with inputs of varying size and composition.
3. Use a function to time it, making sure to only time the algorithm and not set-up time of the Java VM, or parsing input!
4. Plot the results.

Two criteria

Reproducible

- Describe how you obtained instances and measurements in sufficient detail (e.g. which computer, how many runs to compute average?)
- What did you do about initialization, input and output/visualization time?

Two criteria

Reproducible

- Describe how you obtained instances and measurements in sufficient detail (e.g. which computer, how many runs to compute average?)
- What did you do about initialization, input and output/visualization time?

Understandable

- Describe your expectations
- Describe your observations
- Report quality (if not optimal), runtime, and operations count
- Figures/plots for trends (has axes, title, legend) and tables for details
- Use appropriate statistics
    - averages/mean, standard error
    - interpolation: fitting models to data (regression)
    - significant differences (t-test)
    - no millisecond test bed (small problems are solved quickly and runtimes are difficult to compare)

Greedier mode

Something you've seen before

Dijkstra's Algorithm is also greedy. Read the correctness proof in the book and make sure you understand.

Greedier mode

Something you've seen before

Dijkstra's Algorithm is also greedy. Read the correctness proof in the book and make sure you understand.

Next time on Algorithm Design

More greedy algorithms and proving techniques!

- MST (Ch. 4.5-4.6)
- Clustering (Chapter 4.7)

You are here

The course so far

- Course organisation, refreshers, introduction to greedy algorithms

Today's content

- Second proof for "Selecting Breakpoint" (camping sites)
- New problems: Interval Scheduling & Scheduling to Minimize Lateness
- Running experiments

The future

- Greedy (week 2): MSTs & clustering, Huffman encoding
- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

What is still unclear?

Question: After every lecture...

Give us some homework and tell us:
What is still unclear after attending today's lecture?

Homework for this week

- Before next lecture:
  - Do all skills of module Greedy until "Lecture 3" (for your chosen path)
- Next TA check:
  - *Experiments!* (Mountain Climber): November 23
  - *Greedy Triathlon*: November 24 (deadline Nov 25)
- Next peer review:
  - November 23 (during the lab)

CSE2310 Algorithm Design

Lecture 2: Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2