Teoria dos Grafos e Computabilidade

— Greedy algorithms —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.

Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.
- Greedy algorithms: make the **current best choice**.

Teoria dos Grafos e Computabilidade

— Coin change —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Coin change

INSTANCE Let C be a set of coins $\{c_1, c_2, \cdots, c_n\}$ in which c_i means a coin of a specific value and $c_i = c_j$ if $i = j$. Let S be the amount of the change.

SOLUTION The smallest number of coins to achieve the amount S.

Coin change

INSTANCE Let C be a set of coins $\{c_1, c_2, \cdots, c_n\}$ in which c_i means a coin of a specific value and $c_i = c_j$ if $i = j$. Let S be the amount of the change.

SOLUTION The smallest number of coins to achieve the amount S.

Example

- $C = \{1, 2, 6\}$ and $S = 8$

.

.

Coin change

INSTANCE Let C be a set of coins $\{c_1, c_2, \cdots, c_n\}$ in which c_i means a coin of a specific value and $c_i = c_j$ if $i = j$. Let S be the amount of the change.

SOLUTION The smallest number of coins to achieve the amount S.

Example

- $C = \{1, 2, 6\}$ and $S = 8$

What's the smallest number of coins to achieve $S = 8$? .

.

Coin change

INSTANCE Let C be a set of coins $\{c_1, c_2, \cdots, c_n\}$ in which c_i means a coin of a specific value and $c_i = c_j$ if $i = j$. Let S be the amount of the change.

SOLUTION The smallest number of coins to achieve the amount S.

Example

- $C = \{1, 2, 6\}$ and $S = 8$

What's the smallest number of coins to achieve $S = 8$? 2 coins.

.

Coin change

INSTANCE Let C be a set of coins $\{c_1, c_2, \cdots, c_n\}$ in which c_i means a coin of a specific value and $c_i = c_j$ if $i = j$. Let S be the amount of the change.

SOLUTION The smallest number of coins to achieve the amount S.

Example

- $C = \{1, 2, 6\}$ and $S = 8$

What's the smallest number of coins to achieve $S = 8$? 2 coins.

Design an algorithm to compute the smallest number of coins.

Teoria dos Grafos e Computabilidade

— Interval Scheduling —

Silvio Jamil F. Guimarães

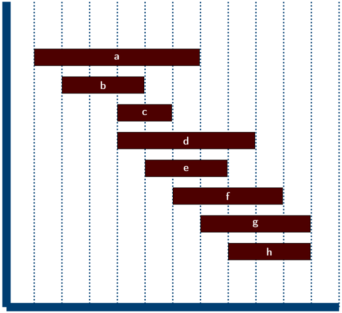
Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Interval Scheduling

INSTANCE Nonempty set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION The largest subset of mutually compatible jobs.

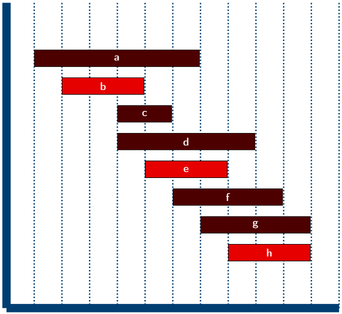


Interval Scheduling

INTERVAL SCHEDULING

INSTANCE Nonempty set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION The largest subset of mutually compatible jobs.



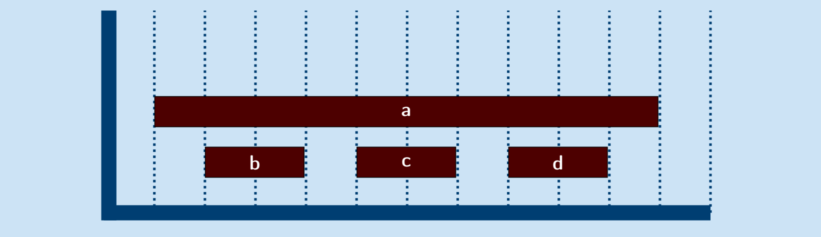
- Two jobs are compatible if they do not overlap.
- This problem models the situation where you have a resource, a set of fixed jobs, and you want to schedule as many jobs as possible.

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

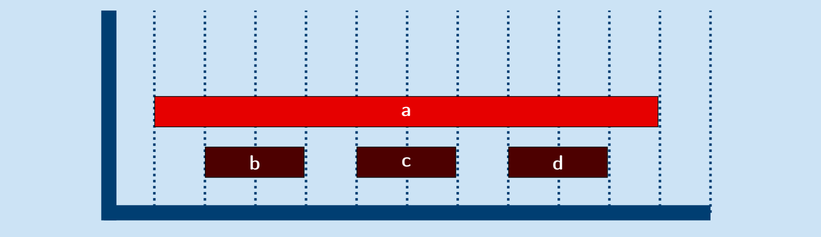


The best solution has **3** compatible jobs. But the it depends on the order in which the jobs are processed !!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

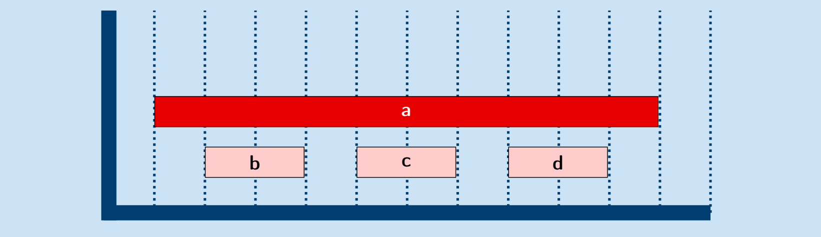
Earliest start time – Increasing order of start time $s(i)$.



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

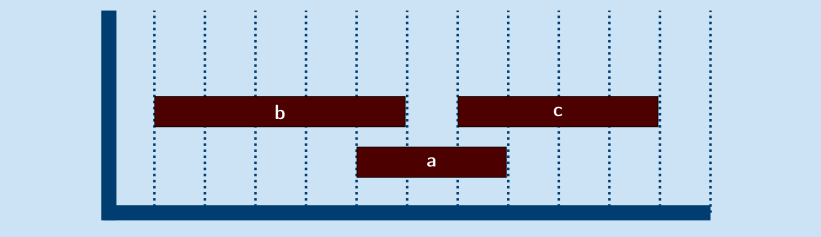
Earliest start time – Increasing order of start time $s(i)$.



The number of compatible jobs using this strategy is **1**, against **3** jobs in the best solution!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

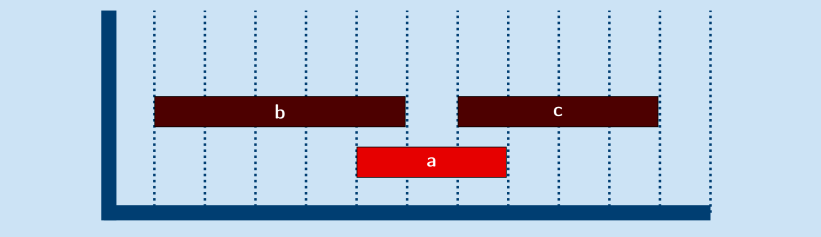


The best solution has **2** compatible jobs. But the it depends on the order in which the jobs are processed !!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

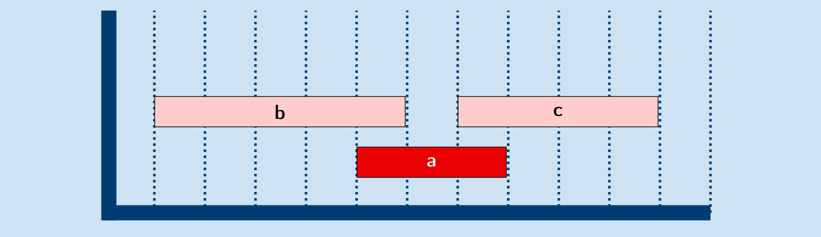
Shortest interval – Increasing order of length $f(i) - s(i)$.



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

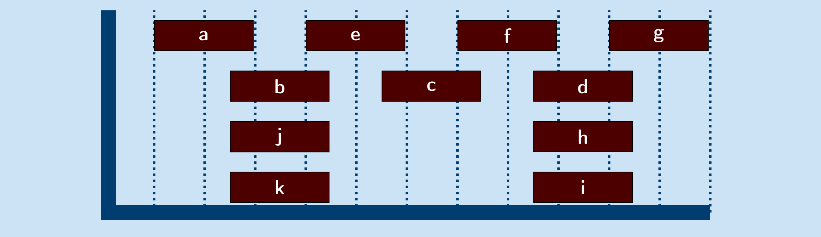
Shortest interval – Increasing order of length $f(i) - s(i)$.



The number of compatible jobs using this strategy is **1**, against **2** jobs in the best solution!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

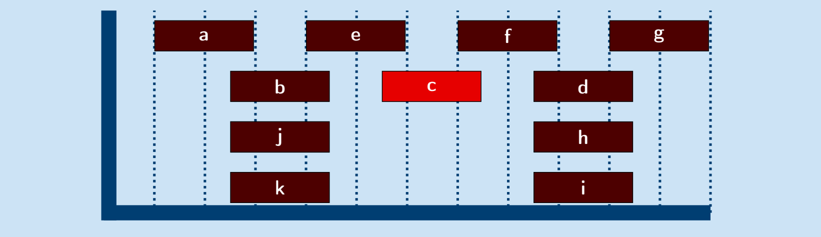


The best solution has **4** compatible jobs. But the it depends on the order in which the jobs are processed !!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

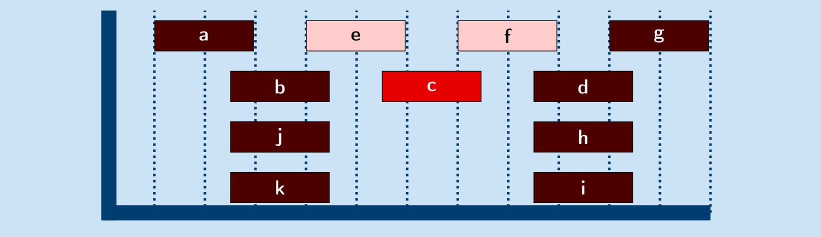
Fewest conflicts – Increasing order of the number of conflicting jobs



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

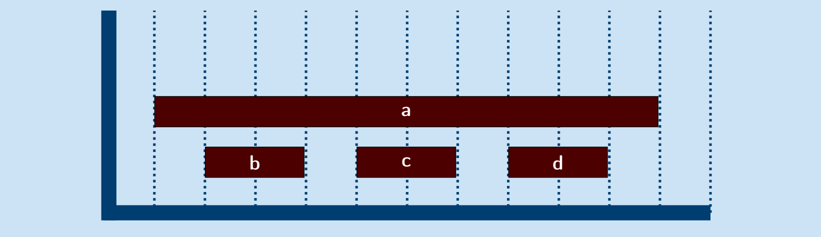
Fewest conflicts – Increasing order of the number of conflicting jobs



The number of compatible jobs using this strategy is **3**, against **4** jobs in the best solution!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

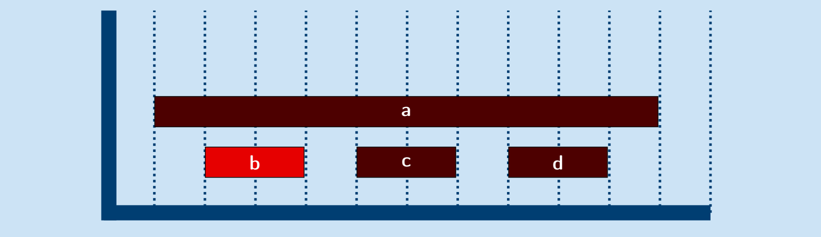


The best solution has **3** compatible jobs. But the it depends on the order in which the jobs are processed !!!!

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

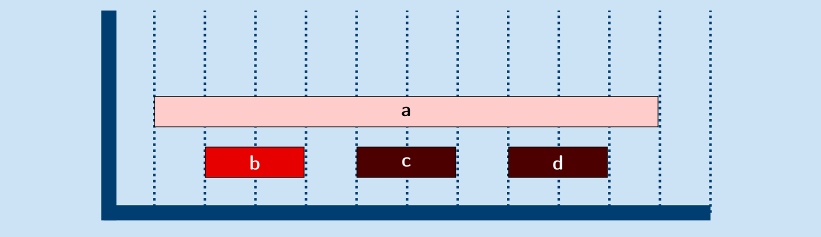
Earliest finish time – Increasing order of finish time $f(i)$.



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

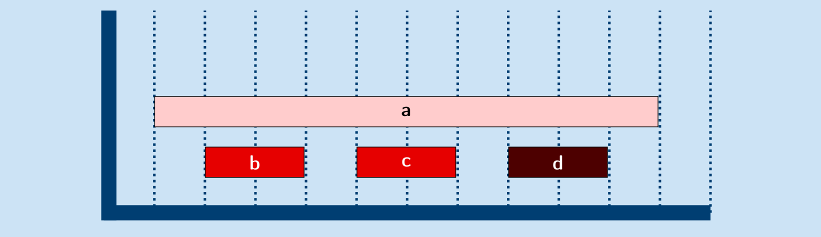
Earliest finish time – Increasing order of finish time $f(i)$.



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

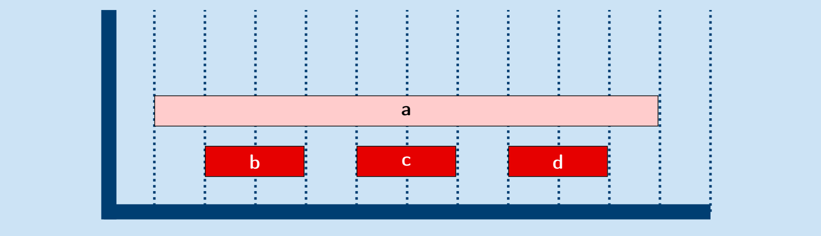
Earliest finish time – Increasing order of finish time $f(i)$.



Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what **order** should we process the jobs?

Earliest finish time – Increasing order of finish time $f(i)$.



The number of compatible jobs using this strategy is **3**.

IS Algorithm: Earliest Finish Time (EFT)

Algorithm: IS Algorithm: Earliest Finish Time (EFT)

input : A set of jobs R
output: A set of compatible jobs A

```
Let R be the set of all jobs;
Let A be an empty set for representing the solution;
while R is not empty do
    Choose a job i ∈ R that has the smallest finishing time;
    Add request i to A;
    Delete all jobs from R that are not compatible with job i;
end
Return the set A as the set of mutually compatible jobs
```

IS Algorithm: Earliest Finish Time (EFT)

Algorithm: IS Algorithm: Earliest Finish Time (EFT)

```
input  : A set of jobs R
output: A set of compatible jobs A

Let R be the set of all jobs;
Let A be an empty set for representing the solution;
while R is not empty do
    Choose a job i ∈ R that has the smallest finishing time;
    Add request i to A;
    Delete all jobs from R that are not compatible with job i;
end
Return the set A as the set of mutually compatible jobs
```

IS Algorithm: Earliest Finish Time (EFT)

Algorithm: IS Algorithm: Earliest Finish Time (EFT)

input : A set of jobs R
output: A set of compatible jobs A

Let R be the set of all jobs;
Let A be an empty set for representing the solution;
while *R is not empty* **do**
    Choose a job $i \in R$ that has the smallest finishing time;
    Add request i to A;
    Delete all jobs from R that are not compatible with job i;
end
Return the set A as the set of mutually compatible jobs

IS Algorithm: Earliest Finish Time (EFT)

Algorithm: IS Algorithm: Earliest Finish Time (EFT)

input : A set of jobs R
output: A set of compatible jobs A

```
Let R be the set of all jobs;
Let A be an empty set for representing the solution;
while R is not empty do
    Choose a job i ∈ R that has the smallest finishing time;
    Add request i to A;
    Delete all jobs from R that are not compatible with job i;
end
Return the set A as the set of mutually compatible jobs
```

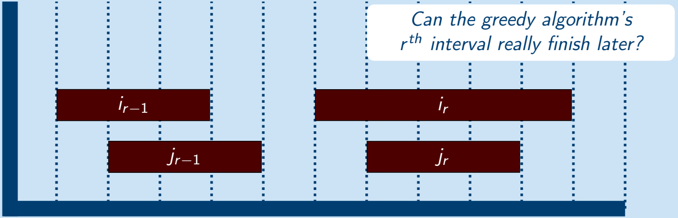
Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in O in order.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in O in order.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

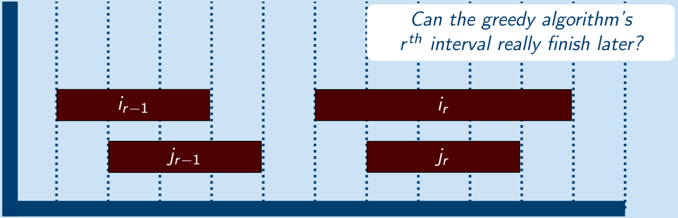
The inductive step in the proof that the greedy algorithm stays ahead



Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in O in order.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

The inductive step in the proof that the greedy algorithm stays ahead



- Claim: The greedy algorithm returns an optimal set A.

Implementing the EFT Algorithm

Reorder jobs so that they are in **increasing order of finish time**.

Store starting time of jobs in an array S.

Implementing the EFT Algorithm

Reorder jobs so that they are in **increasing order of finish time**.

Store starting time of jobs in an array S.

Always select first interval. Let finish time be f.

Implementing the EFT Algorithm

Reorder jobs so that they are in **increasing order of finish time**.

Store starting time of jobs in an array S.

Always select first interval. Let finish time be f.

Iterate over S to find the first index i such that $S[i] \geq f$.

Implementing the EFT Algorithm

Reorder jobs so that they are in **increasing order of finish time**.

Store starting time of jobs in an array S.

Always select first interval. Let finish time be f.

Iterate over S to find the first index i such that $S[i] \geq f$.

Running time is $O(n \log n)$, **dominated by sorting**.

Teoria dos Grafos e Computabilidade

— Interval Partitioning —

Silvio Jamil F. Guimarães

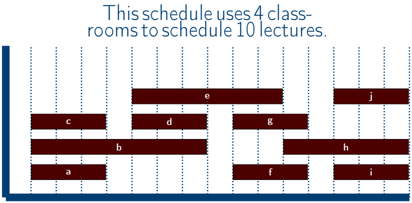
Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Interval Partitioning

Interval Partitioning

INSTANCE Set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION A partition of the jobs into k sets, where each set of jobs is mutually compatible, and k is minimised.

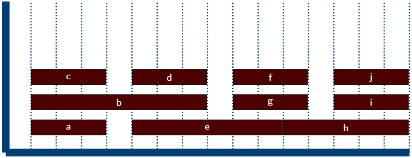


Interval Partitioning

INSTANCE Set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION A partition of the jobs into k sets, where each set of jobs is mutually compatible, and k is minimised.

This schedule uses only 3 classrooms to schedule 10 lectures.



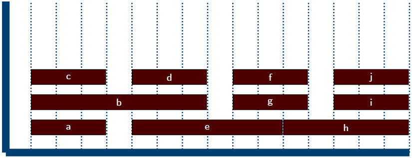
Interval Partitioning

Interval Partitioning

INSTANCE Set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

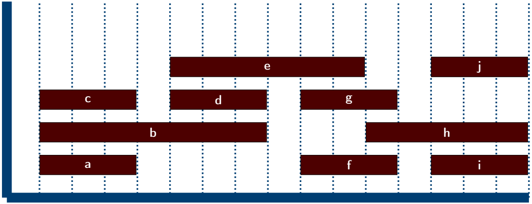
SOLUTION A partition of the jobs into k sets, where each set of jobs is mutually compatible, and k is minimised.

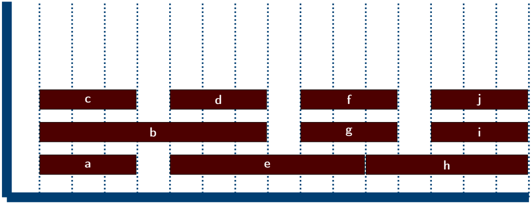
This schedule uses only 3 classrooms to schedule 10 lectures.



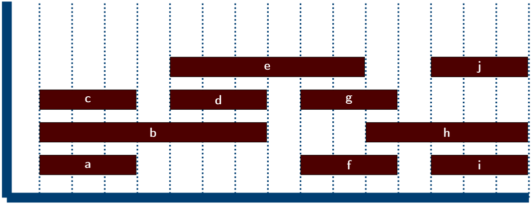
- This problem models the situation where you have set of fixed jobs, and you want to schedule all jobs using as few resources as possible.

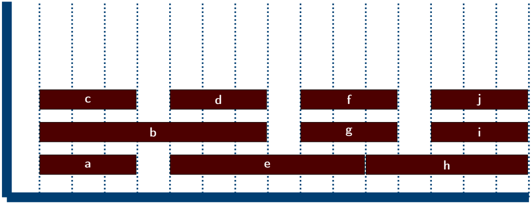
Depth of Intervals





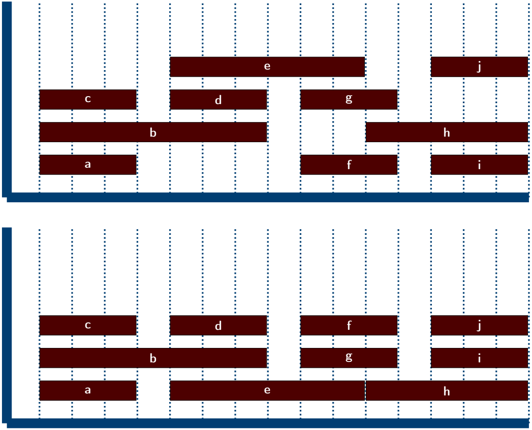
Depth of Intervals





- The **depth** of a set of intervals is the maximum number that contain any time point.

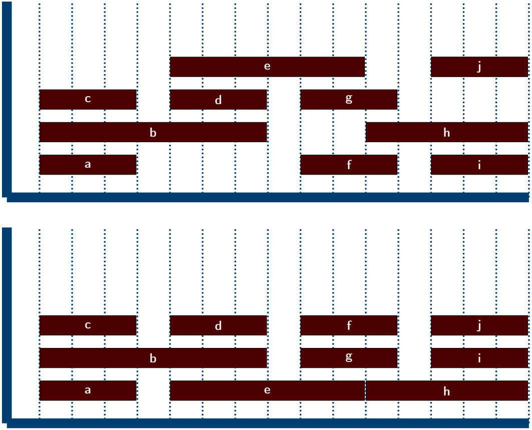
Depth of Intervals



The **depth** is equal to 3

- The **depth** of a set of intervals is the maximum number that contain any time point.

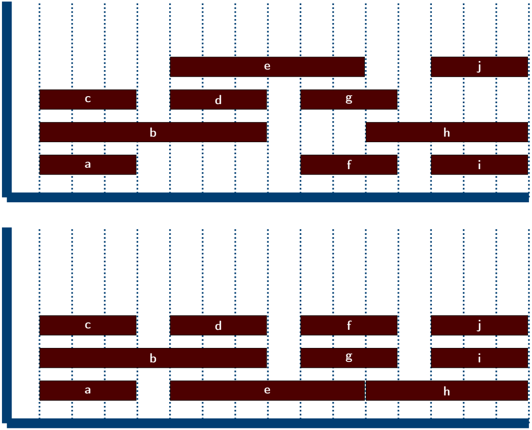
Depth of Intervals



The depth is equal to 3

- The **depth** of a set of intervals is the maximum number that contain any time point.
- Claim: In any instance of INTERVAL PARTITIONING, $k \geq$ depth.

Depth of Intervals



The **depth** is equal to 3

- The **depth** of a set of intervals is the maximum number that contain any time point.
- Claim: In any instance of INTERVAL PARTITIONING, $k \geq$ depth.
- Is it possible to compute k efficiently? Is $k =$ depth?

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R

output: K sets of mutually compatible jobs

```
Sort the interval by their start times, breaking ties arbitrarily;
Let I_1, I_2, ..., I_n, denote the interval in this order;
for j = 1 to n do
    foreach interval I_i that preceds I_j in sorted order and overlaps it do
        Exclude the labels of I_i from consideration for I_j
    end
    if there is any label from {1, 2, ..., d} that has not been excluded then
        Assign a nonexcluded label to I_j
    else
        Leave I_j unlabeled
    end
end
```

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R
output: K sets of mutually compatible jobs

Sort the interval by their start times, breaking ties arbitrarily;
Let $I_1, I_2, \cdots, I_n$, denote the interval in this order;
for $j = 1$ **to** n **do**
    foreach *interval* I_i *that preceds* I_j *in sorted order and overlaps it* **do**
        Exclude the labels of I_i from consideration for I_j
    end
    if *there is any label from* $\{1, 2, \cdots, d\}$ *that has not been excluded* **then**
        Assign a nonexcluded label to I_j
    else
        Leave I_j unlabeled
    end
end

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R

output: K sets of mutually compatible jobs

Sort the interval by their start times, breaking ties arbitrarily;
Let $I_1, I_2, \cdots, I_n$, denote the interval in this order;

for $j = 1$ **to** n **do**
  foreach *interval I_i that preceds I_j in sorted order and overlaps it* **do**
    Exclude the labels of I_i from consideration for I_j
  end
  if *there is any label from $\{1, 2, \cdots, d\}$ that has not been excluded* **then**
    Assign a nonexcluded label to I_j
  else
    Leave I_j unlabeled
  end
end

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R

output: K sets of mutually compatible jobs

Sort the interval by their start times, breaking ties arbitrarily;

Let $I_1, I_2, \cdots, I_n$, denote the interval in this order;

for $j = 1$ **to** n **do**

    foreach *interval I_i that preceds I_j in sorted order and overlaps it* **do**

        Exclude the labels of I_i from consideration for I_j

    end

    if *there is any label from $\{1, 2, \cdots, d\}$ that has not been excluded* **then**

        Assign a nonexcluded label to I_j

    else

        Leave I_j unlabeled

    end

end

- Every interval gets a label and **no pair of overlapping intervals** get the same label.

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R
output: K sets of mutually compatible jobs

```
Sort the interval by their start times, breaking ties arbitrarily;
Let I_1, I_2, ··· , I_n, denote the interval in this order;
for j = 1 to n do
    foreach interval I_i that preceds I_j in sorted order and overlaps it do
        Exclude the labels of I_i from consideration for I_j
    end
    if there is any label from {1, 2, ··· , d} that has not been excluded then
        Assign a nonexcluded label to I_j
    else
        Leave I_j unlabeled
    end
end
```

- Every interval gets a label and **no pair of overlapping intervals** get the same label.

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R
output: K sets of mutually compatible jobs

Sort the interval by their start times, breaking ties arbitrarily;
Let $I_1, I_2, \cdots, I_n$, denote the interval in this order;

for $j = 1$ **to** n **do**
  foreach *interval I_i that preceds I_j in sorted order and overlaps it* **do**
    Exclude the labels of I_i from consideration for I_j
  end
  if *there is any label from $\{1, 2, \cdots, d\}$ that has not been excluded* **then**
    Assign a nonexcluded label to I_j
  else
    Leave I_j unlabeled
  end
end

- Every interval gets a label and **no pair of overlapping intervals** get the same label.
- The greedy algorithm is **optimal**.

Interval Partitioning Algorithm

Algorithm: Interval partitioning algorithm

input : A set of jobs R

output: K sets of mutually compatible jobs

```
Sort the interval by their start times, breaking ties arbitrarily;
Let I_1, I_2, ..., I_n, denote the interval in this order;

for j = 1 to n do
    foreach interval I_i that preceds I_j in sorted order and overlaps it do
        Exclude the labels of I_i from consideration for I_j
    end
    if there is any label from {1, 2, ..., d} that has not been excluded then
        Assign a nonexcluded label to I_j
    else
        Leave I_j unlabeled
    end
end
```

- Every interval gets a label and **no pair of overlapping intervals** get the same label.
- The greedy algorithm is **optimal**.
- The running time of the algorithm is $O(n \log n)$.

Teoria dos Grafos e Computabilidade

— Minimising Lateness —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job i is **delayed** if $f(i) > d(i)$; the **lateness of the job** is $\max(0, f(i) - d(i))$.
- The **lateness of a schedule** is $\max_i \max(0, f(i) - d(i))$.

Scheduling to Minimise Lateness

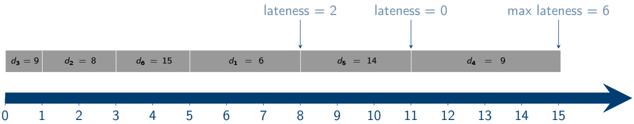
- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job i is **delayed** if $f(i) > d(i)$; the **lateness of the job** is $\max(0, f(i) - d(i))$.
- The **lateness of a schedule** is $\max_i \max(0, f(i) - d(i))$.

	1	2	3	4	5	6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job i is **delayed** if $f(i) > d(i)$; the **lateness of the job** is $\max(0, f(i) - d(i))$.
- The **lateness of a schedule** is $\max_i \max(0, f(i) - d(i))$.

	1	2	3	4	5	6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15



Template for Greedy Algorithm

Minimise Lateness

INSTANCE Set $\{(t(i), d(i)), 1 \leq i \leq n\}$ of lengths and deadlines of n jobs.

SOLUTION Set $\{s(i), 1 \leq i \leq n\}$ of start times such that $\max_i \max(0, s(i) + t(i) - d(i))$ is as small as possible.

Template for Greedy Algorithm

Minimise Lateness

INSTANCE Set $\{(t(i), d(i)), 1 \leq i \leq n\}$ of lengths and deadlines of n jobs.

SOLUTION Set $\{s(i), 1 \leq i \leq n\}$ of start times such that $\max_i \max(0, s(i) + t(i) - d(i))$ is as small as possible.

- Key question: In what order should we schedule the jobs?

Template for Greedy Algorithm

MINIMISE LATENESS

INSTANCE Set $\{(t(i), d(i)), 1 \leq i \leq n\}$ of lengths and deadlines of n jobs.

SOLUTION Set $\{s(i), 1 \leq i \leq n\}$ of start times such that $\max_i \max(0, s(i) + t(i) - d(i))$ is as small as possible.

- Key question: In what order should we schedule the jobs?

  Shortest length *Increasing order of length $t(i)$.*
  Shortest slack time *Increasing order of $d(i) - t(i)$.*
  Earliest deadline *Increasing order of deadline $d(i)$.*

Template for Greedy Algorithm

Shortest length

Increasing order of length $t(i)$.

Template for Greedy Algorithm

Shortest length

Increasing order of length $t(i)$.

	1	2
t_i	1	10
d_i	100	10

counter-example

Template for Greedy Algorithm

Shortest length

Increasing order of length $t(i)$.

	1	2
t_i	1	10
d_i	100	10

counter-example

Shortest slack time

Increasing order of $d(i) - t(i)$.

Template for Greedy Algorithm

Shortest length

Increasing order of length $t(i)$.

	1	2
t_i	1	10
d_i	100	10

counter-example

Shortest slack time

Increasing order of $d(i) - t(i)$.

	1	2
t_i	1	10
d_i	2	10

counter-example

Minimising Lateness: Earliest Deadline First (EDF)

Algorithm: Minimising lateness algorithm

input : A set of jobs R
output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

```
Sort the jobs in order of their deadlines;
Assume, for simplicity, that d_1 ≤ ··· ≤ d_n;
Initially, f = s;
for j = 1 to n do
    Assign the job i to the time interval from s(i) = f to f(i) = f + t_i;
    Let f = f + t_i
end
```

Minimising Lateness: Earliest Deadline First (EDF)

Algorithm: Minimising lateness algorithm

input : A set of jobs R
output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

```
Sort the jobs in order of their deadlines;
Assume, for simplicity, that d_1 ≤ ··· ≤ d_n;
Initially, f = s;
for j = 1 to n do
    Assign the job i to the time interval from s(i) = f to f(i) = f + t_i;
    Let f = f + t_i
end
```

Minimising Lateness: Earliest Deadline First (EDF)

Algorithm: Minimising lateness algorithm

input : A set of jobs R
output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

```
Sort the jobs in order of their deadlines;
Assume, for simplicity, that d_1 ≤ ··· ≤ d_n;
Initially, f = s;
for j = 1 to n do
    Assign the job i to the time interval from s(i) = f to f(i) = f + t_i;
    Let f = f + t_i
end
```

Minimising Lateness: Earliest Deadline First (EDF)

Algorithm: Minimising lateness algorithm

input : A set of jobs R

output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

```
Sort the jobs in order of their deadlines;
Assume, for simplicity, that d_1 ≤ ··· ≤ d_n;
Initially, f = s;
for j = 1 to n do
    Assign the job i to the time interval from s(i) = f to f(i) = f + t_i;
    Let f = f + t_i
end
```

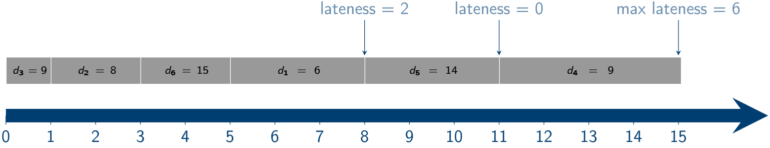
Minimising Lateness: Earliest Deadline First (EDF)

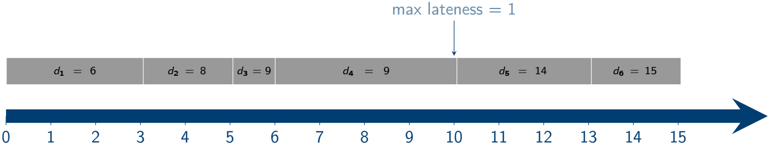
Algorithm: Minimising lateness algorithm

input : A set of jobs R
output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

Sort the jobs in order of their deadlines;
Assume, for simplicity, that $d_1 \leq \cdots \leq d_n$;
Initially, $f = s$;

for $j = 1$ **to** n **do**
    Assign the job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$;
    Let $f = f + t_i$
end





Minimising Lateness: Earliest Deadline First (EDF)

Algorithm: Minimising lateness algorithm

input : A set of jobs R

output: The set of scheduled interval $[s(i), f(i)]$ for $i = 1, \cdots, n$

Sort the jobs in order of their deadlines;
Assume, for simplicity, that $d_1 \leq \cdots \leq d_n$;
Initially, $f = s$;

for $j = 1$ **to** n **do**
  Assign the job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$;
  Let $f = f + t_i$
end

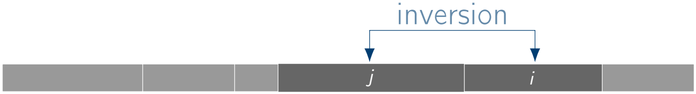
- Proof of correctness is more complex.
- We will use an exchange argument: gradually modify the optimal schedule O till it is the same as the schedule A computed by the algorithm.

Properties of Schedules

- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.

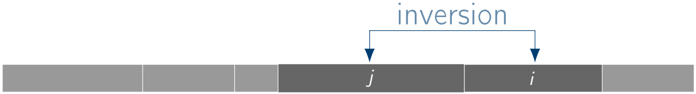
Properties of Schedules

- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



Properties of Schedules

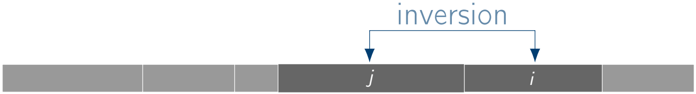
- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



- The algorithm produces a schedule with **no inversions** and no idle time.

Properties of Schedules

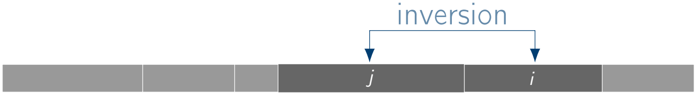
- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



- The algorithm produces a schedule with **no inversions** and no idle time.
- All schedules with no inversions and no idle time have the same lateness.

Properties of Schedules

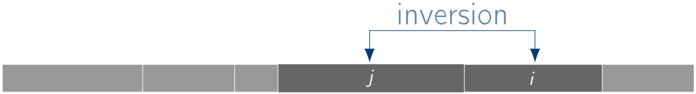
- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



- The algorithm produces a schedule with **no inversions** and no idle time.
- All schedules with no inversions and no idle time have the same lateness.
- There is an optimal schedule with no idle time.

Properties of Schedules

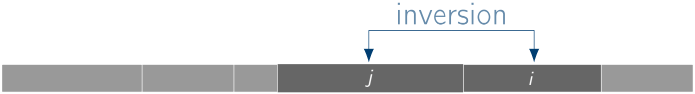
- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



- The algorithm produces a schedule with **no inversions** and no idle time.
- All schedules with no inversions and no idle time have the same lateness.
- There is an optimal schedule with no idle time.
- There is an optimal schedule with no inversions and no idle time.

Properties of Schedules

- A schedule has an **inversion** if a job j with deadline $d(j)$ is scheduled before a job i with an earlier deadline $d(i)$, i.e., $d(i) < d(j)$ and $s(j) < s(i)$.



- The algorithm produces a schedule with **no inversions** and no idle time.
- All schedules with no inversions and no idle time have the same lateness.
- There is an optimal schedule with no idle time.
- There is an optimal schedule with no inversions and no idle time.
- The greedy algorithm produces an **optimal schedule**.

Properties of the Optimal Schedule

- Claim: the optimal schedule O has no inversions and no idle time.
  1. If O has an inversion, then there is a pair of jobs i and j such that i is scheduled just before i and $d(i) < d(j)$.

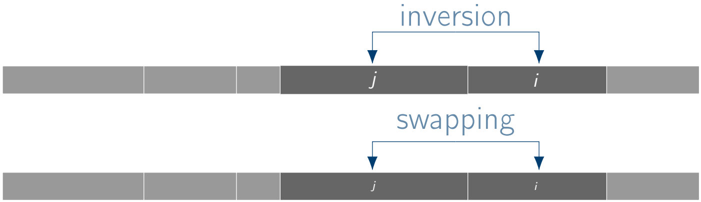
Properties of the Optimal Schedule

- Claim: the optimal schedule O has no inversions and no idle time.
  1. If O has an inversion, then there is a pair of jobs i and j such that i is scheduled just before i and $d(i) < d(j)$.
  2. Let i and j be consecutive inverted jobs in O. After swapping i and j, we get a schedule O' with one less inversion.

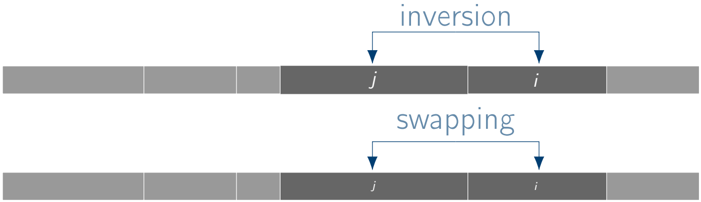
Properties of the Optimal Schedule

- Claim: the optimal schedule O has no inversions and no idle time.
  1. If O has an inversion, then there is a pair of jobs i and j such that i is scheduled just before i and $d(i) < d(j)$.
  2. Let i and j be consecutive inverted jobs in O. After swapping i and j, we get a schedule O' with one less inversion.
  3. The maximum lateness of O' is no larger than the maximum lateness of O.
- If we can prove the last item, we are done, since after $\binom{n}{2}$ swaps, we obtain a schedule with no inversions whose maximum lateness is no larger than that of O.

Swapping Inverted Jobs

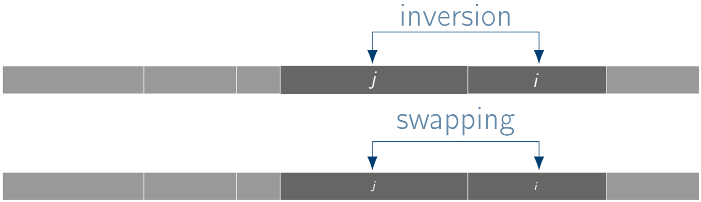


Swapping Inverted Jobs



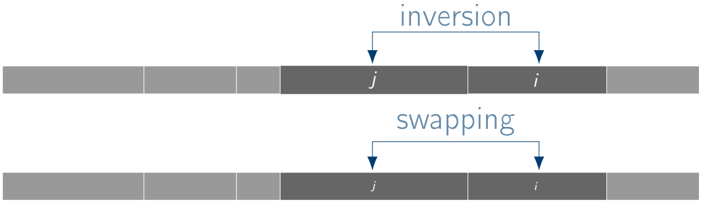
- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness values be $l'(r)$.

Swapping Inverted Jobs



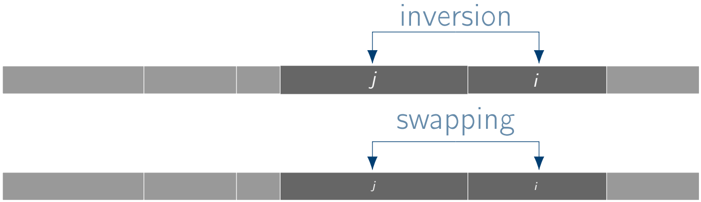
- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness values be $l'(r)$.
- $l'(k) = l(k)$, for all $k \neq i, j$.

Swapping Inverted Jobs



- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness values be $l'(r)$.
- $l'(k) = l(k)$, for all $k \neq i, j$.
- $l'(j) \leq l(j)$.

Swapping Inverted Jobs



- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness values be $l'(r)$.
- $l'(k) = l(k)$, for all $k \neq i, j$.
- $l'(j) \leq l(j)$.
- $l'(i) \leq l(j)$.

Summary

- Greedy algorithms make local decisions.
- Three analysis strategies:

Greedy algorithm stays ahead *Show that After each step in the greedy algorithm, its solution is at least as good as that produced by any other algorithm.*

Structural bound *First, discover a property that must be satisfied by every possible solution. Then show that the (greedy) algorithm produces a solution with this property.*

Exchange argument *Transform the optimal solution in steps into the solution by the greedy algorithm without worsening the quality of the optimal solution.*