Greedy Algorithms

Lecture 20

November 13, 2018

Part I

Greedy Algorithms: Tools and Techniques

What is a Greedy Algorithm?

What is a Greedy Algorithm?

No real consensus on a universal definition.

What is a Greedy Algorithm?

No real consensus on a universal definition.

Greedy algorithms:

1. make decision incrementally in small steps *without backtracking*
2. decision at each step is based on improving *local or current* state in a myopic fashion without paying attention to the *global* situation
3. decisions often based on some fixed and simple *priority* rules

Pros and Cons of Greedy Algorithms

Pros:

1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Pros and Cons of Greedy Algorithms

Pros:

1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Cons:

1. **Very often** greedy algorithms don't work. Easy to lull oneself into believing they work
2. Many greedy algorithms possible for a problem and no structured way to find effective ones

Pros and Cons of Greedy Algorithms

Pros:

1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Cons:

1. **Very often** greedy algorithms don't work. Easy to lull oneself into believing they work
2. Many greedy algorithms possible for a problem and no structured way to find effective ones

CS 374: Every greedy algorithm needs a proof of correctness

Greedy Algorithm Types

Crude classification:

1. Non-adaptive: fix some ordering of decisions a priori and stick with the order
2. Adaptive: make decisions adaptively but greedily/locally at each step

Greedy Algorithm Types

Crude classification:

1. Non-adaptive: fix some ordering of decisions a priori and stick with the order
2. Adaptive: make decisions adaptively but greedily/locally at each step

Plan:

1. See several examples
2. Pick up some proof techniques

Part II

Scheduling Jobs to Minimize Average Waiting Time

The Problem

- n jobs $J_1, J_2, \ldots, J_n$. J_i has non-negative processing time p_i
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average *waiting time*
- Waiting time of J_i in schedule σ: sum of processing times of all jobs scheduled before J_i

	J_1	J_2	J_3	J_4	J_5	J_6
time	3	4	1	8	2	6

The Problem

- n jobs $J_1, J_2, \ldots, J_n$. J_i has non-negative processing time p_i
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average *waiting time*
- Waiting time of J_i in schedule σ: sum of processing times of all jobs scheduled before J_i

	J_1	J_2	J_3	J_4	J_5	J_6
time	3	4	1	8	2	6

Example: schedule is $J_1, J_2, J_3, J_4, J_5, J_6$. Total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =$$

The Problem

- n jobs $J_1, J_2, \ldots, J_n$. J_i has non-negative processing time p_i
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average *waiting time*
- Waiting time of J_i in schedule σ: sum of processing times of all jobs scheduled before J_i

	J_1	J_2	J_3	J_4	J_5	J_6
time	3	4	1	8	2	6

Example: schedule is $J_1, J_2, J_3, J_4, J_5, J_6$. Total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =$$

Optimal schedule:

The Problem

- n jobs $J_1, J_2, \ldots, J_n$. J_i has non-negative processing time p_i
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average *waiting time*
- Waiting time of J_i in schedule σ: sum of processing times of all jobs scheduled before J_i

	J_1	J_2	J_3	J_4	J_5	J_6
time	3	4	1	8	2	6

Example: schedule is $J_1, J_2, J_3, J_4, J_5, J_6$. Total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =$$

Optimal schedule: Shortest Job First. $J_3, J_5, J_1, J_2, J_6, J_4$.

Optimality of SJF

Theorem

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

Optimality of SJF

Theorem

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

Proof strategy: exchange argument

Optimality of SJF

Theorem

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

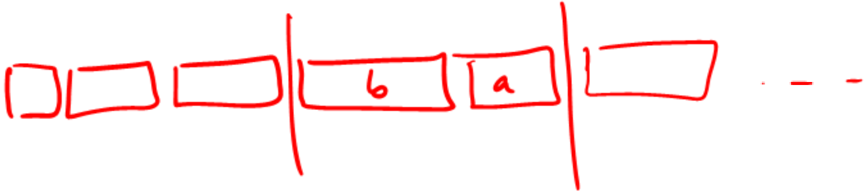
Proof strategy: exchange argument

Assume without loss of generality that job sorted in increasing order of processing time and hence $p_1 \leq p_2 \leq \ldots \leq p_n$ and SJF order is $J_1, J_2, \ldots, J_n$.

Inversions

Definition

A schedule $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ is said to have an inversion if there are jobs J_a and J_b such that S schedules J_a before J_b, but $p_a > p_b$.



Inversions

Definition

A schedule $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ is said to have an inversion if there are jobs J_a and J_b such that S schedules J_a before J_b, but $p_a > p_b$.

Claim

If a schedule has an inversion then there is an inversion between two adjacently *scheduled jobs.*

Proof: exercise.

Proof of optimality of SJF

Recall SJF order is $J_1, J_2, \ldots, J_n$.

- Let $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ be an optimum schedule with fewest inversions.
- If schedule has no inversions then it is identical to SJF schedule and we are done.
- Otherwise there is an $1 \leq \ell < n$ such that $i_\ell > i_{\ell+1}$ since schedule has inversion among two adjacently scheduled jobs

Proof of optimality of SJF

Recall SJF order is $J_1, J_2, \ldots, J_n$.

- Let $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ be an optimum schedule with fewest inversions.
- If schedule has no inversions then it is identical to SJF schedule and we are done.
- Otherwise there is an $1 \leq \ell < n$ such that $i_\ell > i_{\ell+1}$ since schedule has inversion among two adjacently scheduled jobs

Claim

The schedule obtained from $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ by exchanging/swapping positions of jobs J_{i_ℓ} and $J_{i_{\ell+1}}$ is also optimal and has one fewer inversion.

Assuming claim we obtain a contradiction and hence optimum schedule with fewest inversions must be the SJF schedule.

A Weighted Version

- n jobs $J_1, J_2, \ldots, J_n$. J_i has non-negative processing time p_i and a non-negative weight w_i
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average *waiting time*
- Waiting time of J_i in schedule σ: sum of processing times of all jobs scheduled before J_i
- Goal: minimize total *weighted* waiting time.

	J_1	J_2	J_3	J_4	J_5	J_6
time	3	4	1	8	2	6
weight	10	5	2	100	1	1

$w_1 \quad p_1 \qquad w_2 \quad p_2$

$$\begin{array}{r} w_1 p_1 + w_2(p_1+p_2) \\ - \quad w_2 p_2 + w_1(p_1+p_2) \\ \hline w_2 p_1 - w_1 p_2 < 0 \end{array}$$

$$\Rightarrow \quad \frac{p_1}{w_1} < \frac{p_2}{w_2}$$

Part III

Scheduling to Minimize Lateness

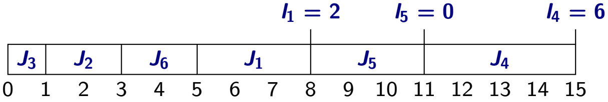
Scheduling to Minimize Lateness

1. Given jobs $J_1, J_2, \ldots, J_n$ with deadlines and processing times to be scheduled on a single resource.
2. If a job i starts at time s_i then it will finish at time $f_i = s_i + t_i$, where t_i is its processing time. d_i: deadline.
3. The lateness of a job is $l_i = \max(0, f_i - d_i)$.
4. Schedule all jobs such that $L = \max l_i$ is minimized.

Scheduling to Minimize Lateness

1. Given jobs $J_1, J_2, \ldots, J_n$ with deadlines and processing times to be scheduled on a single resource.
2. If a job i starts at time s_i then it will finish at time $f_i = s_i + t_i$, where t_i is its processing time. d_i: deadline.
3. The lateness of a job is $l_i = \max(0, f_i - d_i)$.
4. Schedule all jobs such that $L = \max l_i$ is minimized.

	J_1	J_2	J_3	J_4	J_5	J_6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15



Greedy Template

```
Initially R is the set of all requests
curr_time = 0
max_lateness = 0
while R is not empty do
    choose i ∈ R
    curr_time = curr_time + t_i
    if (curr_time > d_i) then
        max_lateness = max(curr_time − d_i, max_lateness)

return max_lateness
```

Main task: Decide the order in which to process jobs in R

Greedy Template

```
Initially R is the set of all requests
curr_time = 0
max_lateness = 0
while R is not empty do
    choose i ∈ R
    curr_time = curr_time + t_i
    if (curr_time > d_i) then
        max_lateness = max(curr_time − d_i, max_lateness)

return max_lateness
```

Main task: Decide the order in which to process jobs in R

Three Algorithms

1. Shortest job first — sort according to t_i.
2. Shortest slack first — sort according to $d_i - t_i$.
3. EDF = Earliest deadline first — sort according to d_i.

Three Algorithms

1. Shortest job first — sort according to t_i.
2. Shortest slack first — sort according to $d_i - t_i$.
3. EDF = Earliest deadline first — sort according to d_i.

Counter examples for first two: exercise

Earliest Deadline First

Theorem

Greedy with EDF *rule minimizes maximum lateness.*

Earliest Deadline First

Theorem

Greedy with EDF *rule minimizes maximum lateness.*

Proof via an exchange argument.

Earliest Deadline First

Theorem

Greedy with EDF *rule minimizes maximum lateness.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

Earliest Deadline First

Theorem

Greedy with EDF *rule minimizes maximum lateness.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

Lemma

If there is a feasible schedule then there is one with no idle time before all jobs are finished.

Inversions

Assume jobs are sorted such that $d_1 \leq d_2 \leq \ldots \leq d_n$. Hence EDF schedules them in this order.

Definition

A schedule S is said to have an inversion if there are jobs i and j such that S schedules i before j, but $d_i > d_j$.

Inversions

Assume jobs are sorted such that $d_1 \leq d_2 \leq \ldots \leq d_n$. Hence EDF schedules them in this order.

Definition

A schedule S is said to have an inversion if there are jobs i and j such that S schedules i before j, but $d_i > d_j$.

Claim

If a schedule S has an inversion then there is an inversion between two adjacently *scheduled jobs.*

Proof: exercise.

Proof sketch of Optimality of EDP

- Let S be an optimum schedule with smallest number of inversions.
- If S has no inversions then this is same as EDF and we are done.
- Else S has two adjacent jobs i and j with $d_i > d_j$.
- Swap positions of i and j to obtain a new schedule S'

Claim

Maximum lateness of S' is no more than that of S. And S' has strictly fewer inversions than S.

$d_2 > d_1$

$\max\{0, d_2 - p_2, d_1 - (p_1 + p_2)\}$

Part IV

Maximum Weight Subset of Elements: Cardinality and Beyond

Picking k elements to maximize total weight

1. Given n items each with non-negative weights/profits and integer $1 \leq k \leq n$.
2. Goal: pick k elements to maximize total weight of items picked.

	e_1	e_2	e_3	e_4	e_5	e_6
weight	3	2	1	4	3	2

$k = 2$:
$k = 3$:
$k = 4$:

Greedy Template

```
N is the set of all elements X ← ∅
(* X will store all the elements that will be picked *)
while |X| < k and N is not empty do
    choose e_j ∈ N of maximum weight
    add e_j to X
    remove e_j from N
return the set X
```

Remark: One can rephrase algorithm simply as sorting elements in decreasing weight order and picking the top k elements but the above template generalizes to other settings a bit more easily.

Greedy Template

```
N is the set of all elements X ← ∅
(* X will store all the elements that will be picked *)
while |X| < k and N is not empty do
    choose e_j ∈ N of maximum weight
    add e_j to X
    remove e_j from N
return the set X
```

Remark: One can rephrase algorithm simply as sorting elements in decreasing weight order and picking the top k elements but the above template generalizes to other settings a bit more easily.

Theorem

Greedy is optimal for picking k elements of maximum weight.

A more interesting problem

1. Given n items $N = \{e_1, e_2, \ldots, e_n\}$. Each item e_i has a non-negative weight w_i.
2. Items partitioned into h sets $N_1, N_2, \ldots, N_h$. Think of each item having one of h colors.
3. Given integers $k_1, k_2, \ldots, k_h$ and another integer k
4. Goal: pick k elements such that no more than k_i from N_i to maximize total weight of items picked.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
weight	9	5	4	7	5	2	1

$N_1 = \{e_1, e_2, e_3\}$, $N_2 = \{e_4, e_5\}$, $N_3 = \{e_6, e_7\}$
$k = 4$, $k_1 = 2, k_2 = 1, k_3 = 2$

Greedy Template

N is the set of all elements $X \leftarrow \emptyset$
(* X will store all the elements that will be picked *)
while N is not empty **do**
    $N' = \{e_i \in N \mid X \cup \{e_i\}$ is feasible$\}$
    If $N' \leftarrow \emptyset$ break
    choose $e_j \in N'$ of maximum weight
    add e_j to X
    remove e_j from N
return the set X

Greedy Template

N is the set of all elements $X \leftarrow \emptyset$
(* X will store all the elements that will be picked *)
while N is not empty **do**
    $N' = \{e_i \in N \mid X \cup \{e_i\}$ is feasible$\}$
    If $N' \leftarrow \emptyset$ break
    choose $e_j \in N'$ of maximum weight
    add e_j to X
    remove e_j from N
return the set X

Theorem

Greedy is optimal for the problem on previous slide.

Proof: exercise after class.

Special case of the general phenomenon of Greedy working for maximum weight indepedent set in a matroid. Beyond scope of

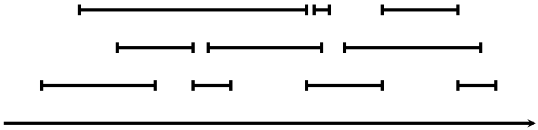
Part V

Interval Scheduling

Interval Scheduling

Problem (Interval Scheduling)

Input: *A set of jobs with start and finish times to be scheduled on a resource (example: classes and class rooms).*
Goal: *Schedule as many jobs as possible*



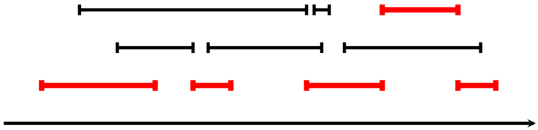
Interval Scheduling

Problem (Interval Scheduling)

Input: *A set of jobs with start and finish times to be scheduled on a resource (example: classes and class rooms).*

Goal: *Schedule as many jobs as possible*

1. *Two jobs with overlapping intervals cannot both be scheduled!*



Greedy Template

```
R is the set of all requests
X ← ∅ (* X will store all the jobs that will be scheduled *)
while R is not empty do
    choose i ∈ R
    add i to X
    remove from R all requests that overlap with i
return the set X
```

Greedy Template

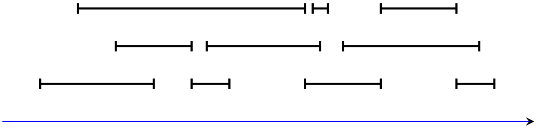
```
R is the set of all requests
X ← ∅ (* X will store all the jobs that will be scheduled *)
while R is not empty do
    choose i ∈ R
    add i to X
    remove from R all requests that overlap with i
return the set X
```

Main task: Decide the order in which to process requests in R

ES SP FC EF

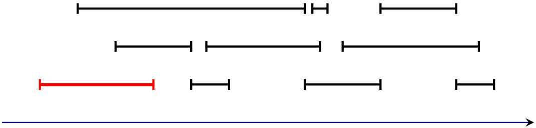
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



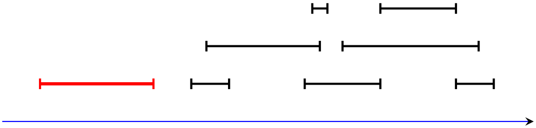
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



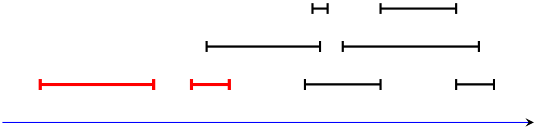
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



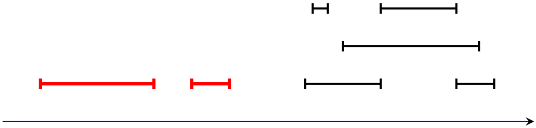
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



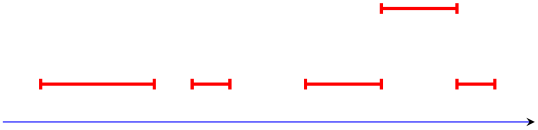
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.



Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

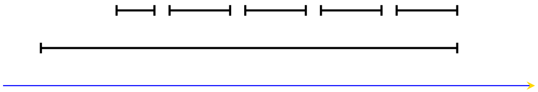


Figure: Counter example for earliest start time

Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

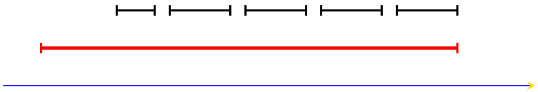


Figure: Counter example for earliest start time

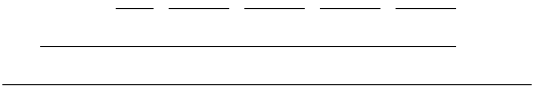
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

Figure: Counter example for earliest start time

Smallest Processing Time

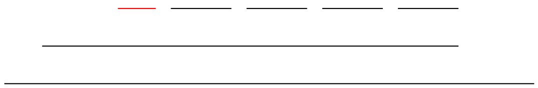
Process jobs in the order of processing time, starting with jobs that require the shortest processing.



Back Counter

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.



Back Counter

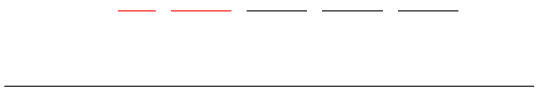
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Back Counter

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.



Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Back Counter

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

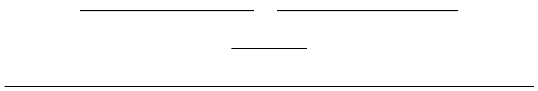


Figure: Counter example for smallest processing time

Back Counter

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

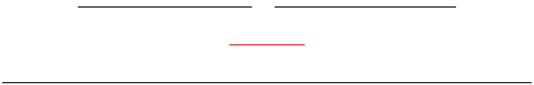


Figure: Counter example for smallest processing time

Back Counter

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

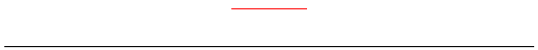
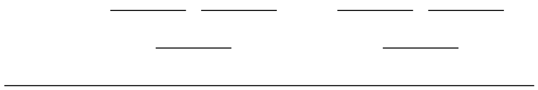


Figure: Counter example for smallest processing time

Back Counter

Fewest Conflicts

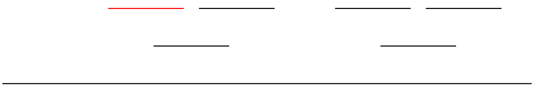
Process jobs in that have the fewest "conflicts" first.



Back Counter

Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.



Back Counter

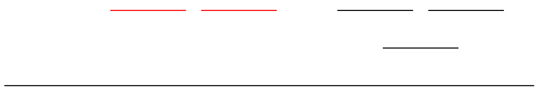
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

Back Counter

Fewest Conflicts

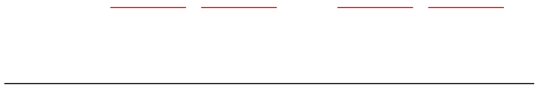
Process jobs in that have the fewest “conflicts” first.



Back Counter

Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.



Back Counter

Fewest Conflicts

Process jobs in that have the fewest "conflicts" first.

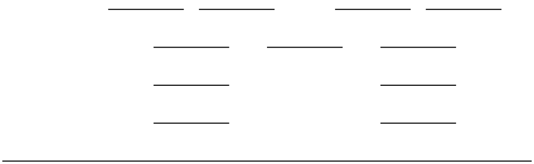


Figure: Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process jobs in that have the fewest "conflicts" first.

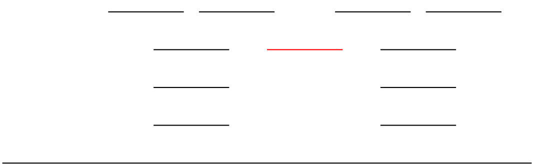


Figure: Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

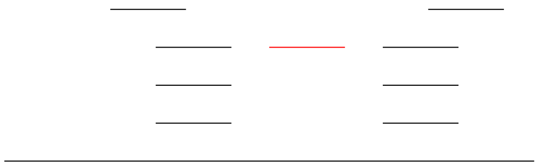


Figure: Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

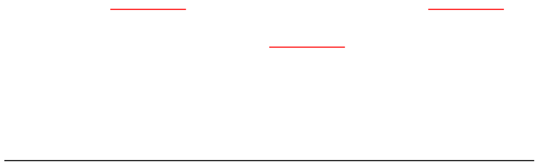
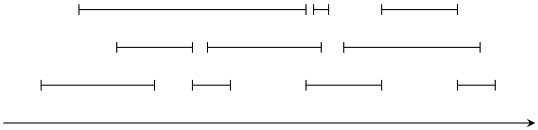


Figure: Counter example for fewest conflicts

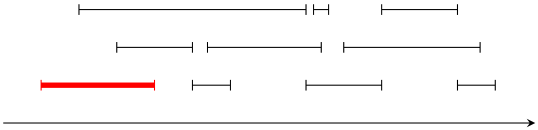
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



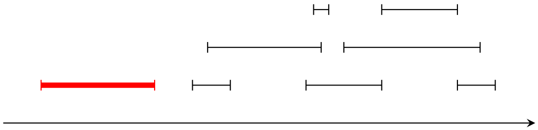
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



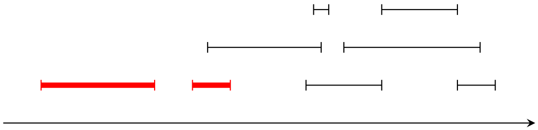
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



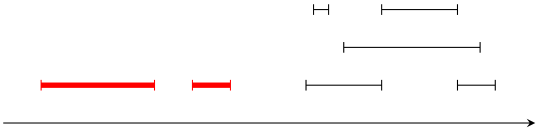
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



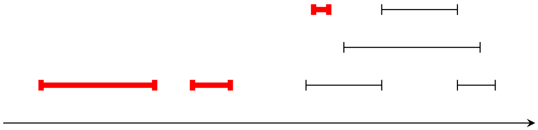
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



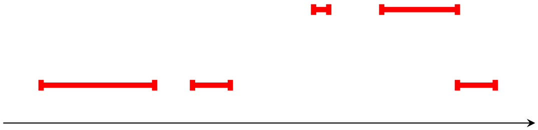
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.



Optimal Greedy Algorithm

R is the set of all requests
$X \leftarrow \emptyset$ (* X stores the jobs that will be scheduled *)
while R is not empty
    choose $i \in R$ such that finishing time of i is smallest
    add i to X
    remove from R all requests that overlap with i
return X

Theorem

The greedy algorithm that picks jobs in the order of their finishing times is optimal.

Proving Optimality

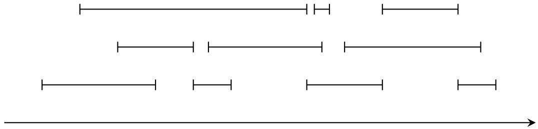
1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts

Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts
2. For a set of requests R, let O be an optimal set and let X be the set returned by the greedy algorithm. Then $O = X$?

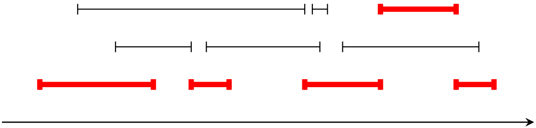
Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts
2. For a set of requests R, let O be an optimal set and let X be the set returned by the greedy algorithm. Then $O = X$?Not likely!



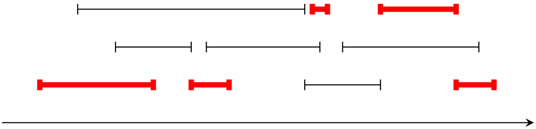
Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts
2. For a set of requests $\mathcal{R}$, let $\mathcal{O}$ be an optimal set and let X be the set returned by the greedy algorithm. Then $\mathcal{O} = X$?Not likely!



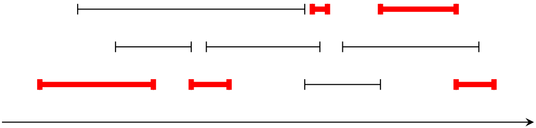
Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts
2. For a set of requests R, let O be an optimal set and let X be the set returned by the greedy algorithm. Then $O = X$?Not likely!



Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts
2. For a set of requests R, let O be an optimal set and let X be the set returned by the greedy algorithm. Then $O = X$?Not likely!



Instead we will show that $|O| = |X|$

Proof of Optimality: Key Lemma

Lemma

Let i_1 be first interval picked by Greedy. There exists an optimum solution that contains i_1.

Proof.

Let O be an *arbitrary* optimum solution. If $i_1 \in O$ we are done.

Proof of Optimality: Key Lemma

Lemma

Let i_1 be first interval picked by Greedy. There exists an optimum solution that contains i_1.

Proof.

Let O be an *arbitrary* optimum solution. If $i_1 \in O$ we are done.
Claim: If $i_1 \notin O$ then there is exactly one interval $j_1 \in O$ that conflicts with i_1. (proof later)

Proof of Optimality: Key Lemma

Lemma

Let i_1 be first interval picked by Greedy. There exists an optimum solution that contains i_1.

Proof.

Let O be an *arbitrary* optimum solution. If $i_1 \in O$ we are done.
Claim: If $i_1 \not\in O$ then there is exactly one interval $j_1 \in O$ that conflicts with i_1. (proof later)

1. Form a new set O' by removing j_1 from O and adding i_1, that is $O' = (O - \{j_1\}) \cup \{i_1\}$.
2. From claim, O' is a *feasible* solution (no conflicts).
3. Since $|O'| = |O|$, O' is also an optimum solution and it contains i_1. □

Proof of Claim

Claim

If $i_1 \not\in O$, there is exactly one interval $j_1 \in O$ that conflicts with i_1.

Proof.

1. If no $j \in O$ conflicts with i_1 then O is not optimal!
2. Suppose $j_1, j_2 \in O$ such that $j_1 \neq j_2$ and both j_1 and j_2 conflict with i_1.
3. Since i_1 has earliest finish time, j_1 and i_1 overlap at $f(i_1)$.
4. For same reason j_2 also overlaps with i_1 at $f(i_1)$.
5. Implies that j_1, j_2 overlap at $f(i_1)$ but intervals in O cannot overlap.

See figure in next slide. □

Figure for proof of Claim

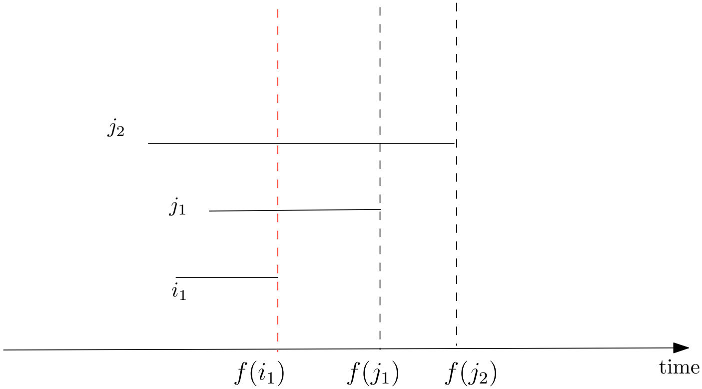


Figure: Since i_1 has the earliest finish time, any interval that conflicts with it does so at $f(i_1)$. This implies j_1 and j_2 conflict.

Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

Base Case: $n = 1$. Trivial since Greedy picks one interval.
Induction Step: Assume theorem holds for $i < n$.
Let I be an instance with n intervals
I': I with i_1 and all intervals that overlap with i_1 removed
$G(I), G(I')$: Solution produced by Greedy on I and I'
From Lemma, there is an optimum solution O to I and $i_1 \in O$.
Let $O' = O - \{i_1\}$. O' is a solution to I'.

$$\begin{aligned} |G(I)| &= 1 + |G(I')| \quad \text{(from Greedy description)} \\ &\geq 1 + |O'| \quad \text{(By induction, } G(I') \text{ is optimum for } I'\text{)} \\ &= |O| \end{aligned}$$

□

Implementation and Running Time

```
Initially R is the set of all requests
X ← ∅ (* X stores the jobs that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    if i does not overlap with requests in X
        add i to X
    remove i from R
return the set X
```

- Presort all requests based on finishing time. $O(n \log n)$ time
- Now choosing least finishing time is $O(1)$
- Keep track of the finishing time of the last request added to A. Then check if starting time of i later than that
- Thus, checking non-overlapping is $O(1)$
- Total time $O(n \log n + n) = O(n \log n)$

Comments

1. Interesting Exercise: smallest interval first picks at least half the optimum number of intervals.
2. All requests need not be known at the beginning. Such *online* algorithms are a subject of research

Weighted Interval Scheduling

Suppose we are given n jobs. Each job i has a start time s_i, a finish time f_i, and a weight w_i. We would like to find a set S of compatible jobs whose total weight is maximized. Which of the following greedy algorithms finds the optimum schedule?

- **(A)** Earliest start time first.
- **(B)** Earliest finish time fist.
- **(C)** Highest weight first.
- **(D)** None of the above.
- **(E)** IDK.

Weighted Interval Scheduling

Suppose we are given n jobs. Each job i has a start time s_i, a finish time f_i, and a weight w_i. We would like to find a set S of compatible jobs whose total weight is maximized. Which of the following greedy algorithms finds the optimum schedule?

- **(A)** Earliest start time first.
- **(B)** Earliest finish time fist.
- **(C)** Highest weight first.
- **(D)** None of the above.
- **(E)** IDK.

Weighted problem can be solved via dynamic prog. See notes.

Greedy Analysis: Overview

1. Greedy's first step leads to an optimum solution. Show that there is an optimum solution leading from the first step of Greedy and then use induction. Example, Interval Scheduling.
2. Greedy algorithm stays ahead. Show that after each step the solution of the greedy algorithm is at least as good as the solution of any other algorithm. Example, Interval scheduling.
3. Structural property of solution. Observe some structural bound of every solution to the problem, and show that greedy algorithm achieves this bound. Example, Interval Partitioning (see Kleinberg-Tardos book).
4. Exchange argument. Gradually transform any optimal solution to the one produced by the greedy algorithm, without hurting its optimality. Example, Minimizing lateness.

Takeaway Points

1. Greedy algorithms come naturally but often are incorrect. A proof of correctness is an absolute necessity.
2. *Exchange* arguments are often the key proof ingredient. Focus on why the first step of the algorithm is correct: need to show that there is an optimum/correct solution with the first step of the algorithm.
3. Thinking about correctness is also a good way to figure out which of the many greedy strategies is likely to work.