Announcements

- HW1 has been due yesterday at 6 pm

- HW2 will be out later today
  - 2 weeks for completing it
  - About randomized algorithms and DP

- Fellow student Yasunari built a Discord Server to facilitate class discussion, study groups, knowing your classmates, etc.
  - Link: https://discord.gg/JPp8gMZ
  - Also see his piazza post: https://piazza.com/class/ke5yn1pnhgm61l?cid=16

CS6161: Design and Analysis of Algorithms (Fall 2020)

Greedy Algorithms

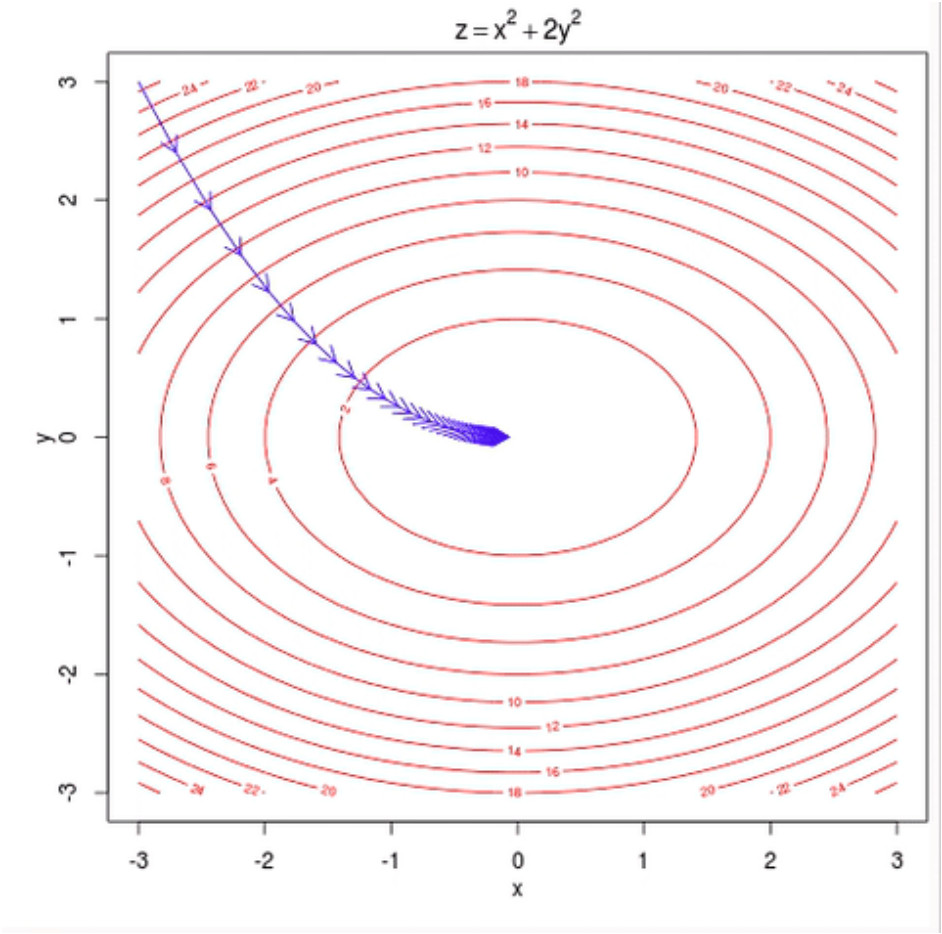
Instructor: Haifeng Xu

Outline

- Greedy Algorithms and a Simple Example
- Greedy Algorithm Analysis for a Non-trivial Problem

What is a Greedy Algorithm?

- Build up a solution by greedily improve local or current state, and hope that it will lead to a globally optimal solution



What is a Greedy Algorithm?

Pros:

- Easy to implement and often run fast
- Can be tried pretty much for any problem
  - Often leads to first-cut heuristic when problems are not well understood
- If works, typically imply useful structures of the problem

> "Greed … is good. Greed is right. Greed works."
>
> -- Michael Douglas, *Wall Street* (1987)

Unfortunately, not true in algorithm design …

What is a Greedy Algorithm?

Pros:

- Easy to implement and often run fast
- Can be tried pretty much for any problem
  - Often leads to first-cut heuristic when problems are not well understood
- If works, typically imply useful structures of the problem

Cons:

- **Very often**, greedy algorithms do not work
- Easy to invent many versions of greedy algorithms for a problem, but finding the right one and proving its correctness is not easy

Unlike DP and D-and-C, greedy algorithms require a proof of correctness in most cases

A Warm-up Example

- A worker has total work time T
- There are n jobs to be completed
  - Job i takes time t_i to complete and results in payoff r_i
  - May complete any α fraction of job i, taking αt_i time with payoff αr_i

Algorithmic Question

Compute the fraction α_i of job i for all i, to maximize payoff

Is this a knapsack problem?

- Not exactly, in knapsack cannot pack 0.5 fraction of an item
- It's called fractional knapsack, and is much easier

Fractional Knapsack – Greed Works

Now you can complete any fraction of a job and getting payoff proportional to it, which job is your most profitable choice?

- The one with maximum payoff r_i?
- The one with shortest time t_i?
- Should be the one with largest average payoff $\frac{r_i}{t_i}$!

Fact: within time t where $t \leq t_i$ for i, doing job i^* with maximum $\frac{r_i}{t_i}$ achieves the largest payoff.

Proof: Payoff of completing job i is

$$\frac{t}{t_i} \times r_i$$

fraction of job i completed

Fractional Knapsack – Greed Works

Now you can complete any fraction of a job and getting payoff proportional to it, which job is your most profitable choice?

- The one with maximum payoff r_i?
- The one with shortest time t_i?
- Should be the one with largest average payoff $\frac{r_i}{t_i}$!

Fact: within time t where $t \leq t_i$ for i, doing job i^* with maximum $\frac{r_i}{t_i}$ achieves the largest payoff.

Proof: Payoff of completing job i is

$$\frac{t}{t_i} \times r_i = t \times \frac{r_i}{t_i}$$

$\frac{t}{t_i}$: fraction of job i completed

Fractional Knapsack – Greed Works

Now you can complete any fraction of a job and getting payoff proportional to it, which job is your most profitable choice?

- The one with maximum payoff r_i?
- The one with shortest time t_i?
- Should be the one with largest average payoff $\frac{r_i}{t_i}$!

Fact: within time t where $t \leq t_i$ for i, doing job i^* with maximum $\frac{r_i}{t_i}$ achieves the largest payoff.

Proof: Payoff of completing job i is

$$\frac{t}{t_i} \times r_i = t \times \frac{r_i}{t_i}$$ which is maximized at i^*

fraction of job i completed

Fractional Knapsack – Greed Works

Now you can complete any fraction of a job and getting payoff proportional to it, which job is your most profitable choice?

- The one with maximum payoff r_i?
- The one with shortest time t_i?
- Should be the one with largest average payoff $\frac{r_i}{t_i}$!

Fact: within time t where $t \leq t_i$ for i, doing job i^* with maximum $\frac{r_i}{t_i}$ achieves the largest payoff.

Remark: this fact is the reason that Greed works.

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^{n})$

1

2

3

4

5

6

7

8

9

10

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^{n})$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2

3

4

5

6

7

8

9

10

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^{n})$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3

4

5

6

7

8

9

10

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3 **for** $i = 1, \cdots, n$ in the sorted order

4 **if** $t > t_i$

5 $R = R + r_i$

6 $t = t - t_i$

7

8

9

10

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3 **for** $i = 1, \cdots, n$ in the sorted order

4 **if** $t > t_i$

5 $R = R + r_i$

6 $t = t - t_i$

7 **else**

8 $R = R + r_i \times \frac{t}{t_i}$

9 **return** R

10

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3 **for** $i = 1, \cdots, n$ in the sorted order

4 **if** $t > t_i$

5 $R = R + r_i$

6 $t = t - t_i$

7 **else**

8 $R = R + r_i \times \frac{t}{t_i}$

9 **return** R

10 **return** R

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3 **for** $i = 1, \cdots, n$ in the sorted order

4 **if** $t > t_i$

5 $R = R + r_i$

6 $t = t - t_i$

7 **else**

8 $R = R + r_i \times \frac{t}{t_i}$

9 **return** R

10 **return** R

$O(n \log n)$ time due to sorting

Fractional Knapsack: Pseudo-Code

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

1 Sort jobs based on $\frac{r_i}{t_i}$ with descending order

2 $t = T; R = 0$ // t: total remaining time; R: total rewards so far

3 **for** $i = 1, \cdots, n$ in the sorted order

4 **if** $t > t_i$

5 $R = R + r_i$

6 $t = t - t_i$

7 **else**

8 $R = R + r_i \times \frac{t}{t_i}$

9 **return** R

10 **return** R

$O(n \log n)$ time due to sorting

Remark: there is a better $O(n)$ implementation using the *median of median algorithm* (we did not cover though) that can avoid sorting. Think about it if you are interested!

Correctness is Relatively Easy Here

Claim: at any time t, the collected reward so far is the largest possible.

Follows from previous fact, as we are completing the jobs in descending order of $\frac{r_i}{t_i}$.

FractionalKnapsack-Greed$(T, \{t_i, r_i\}_{i=1}^n)$

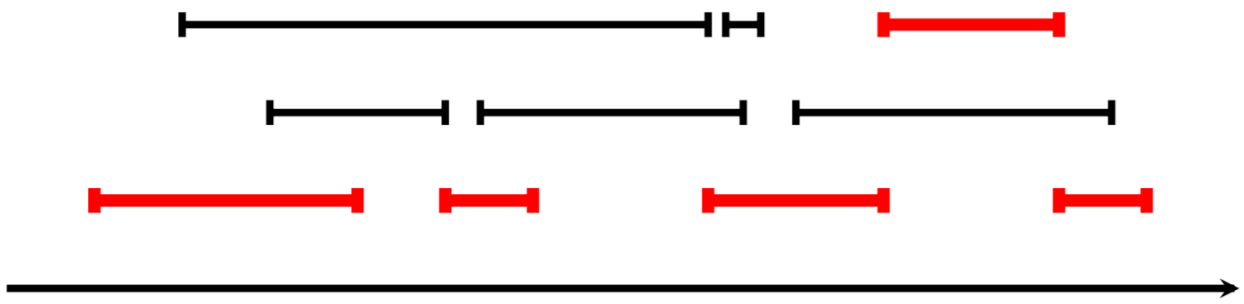
```
1    Sort jobs based on r_i/t_i with descending order
2    t = T; R = 0
3    for i = 1, ..., n in the sorted order
4        if t > t_i
5            R = R + r_i
6            t = t − t_i
7        else
8            R = R + r_i × t/t_i
9            return R
10   return R
```

Outline

- Greedy Algorithms and a Simple Example
- Greedy Algorithm Analysis for a Non-trivial Problem

Example 2: Interval Job Scheduling

- **Input**: a set of jobs, job i starts at s_i and finishes at f_i
  - Two jobs are compatible if they do not overlap
- **Output**: find a maximum subset of mutually compatible jobs



The Greedy Framework

- Okay, so we know it's going to be some greedy algorithm…

SomeGreedyAlgo($\{s_i, f_i\}_{i=1}^n$)

1

2

3

4

5

6

7

The Greedy Framework

- Okay, so we know it's going to be some greedy algorithm…

```
SomeGreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R be the set of all jobs;
2    Initialize X = 0    // X will store all scheduled jobs
3
4
5
6
7
```

The Greedy Framework

- Okay, so we know it's going to be some greedy algorithm…

```
SomeGreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R be the set of all jobs;
2    Initialize X = 0    // X will store all scheduled jobs
3    while R is not empty
4        choose some i ∈ R;
5        add i to X
6
7
```

The Greedy Framework

- Okay, so we know it's going to be some greedy algorithm…

```
SomeGreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R be the set of all jobs;
2    Initialize X = 0    // X will store all scheduled jobs
3    while R is not empty
4        choose some i ∈ R;
5        add i to X
6        remove from R all jobs that overlap with i
7    return X
```

The Greedy Framework

- Okay, so we know it's going to be some greedy algorithm…

```
SomeGreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R be the set of all jobs;
2    Initialize X = 0    // X will store all scheduled jobs
3    while R is not empty
4        choose some i ∈ R;
5        add i to X
6        remove from R all jobs that overlap with i
7    return X
```

Tricky part: decide which job from R to process

The Greedy Framework

Candidate choice of next $i \in R$

- Earliest start time: the $i \in R$ with smallest s_i
- Earliest finish time: the $i \in R$ with smallest f_i
- Shortest interval time: the $i \in R$ with smallest $f_i - s_i$
- Fewest conflicts: the $i \in R$ with minimum #conflicts with other jobs

Does any of these work, and if any, which one?

Does Earliest Start Time Work?

- No

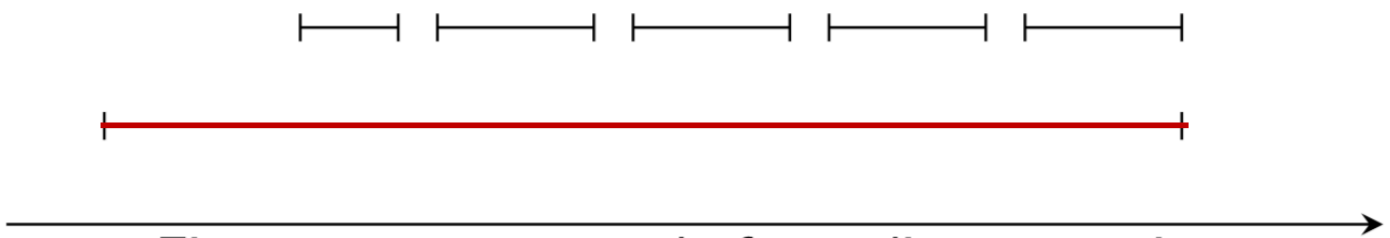


Figure: counterexample for earliest start time

Does Shortest Interval Choice Work?

- No

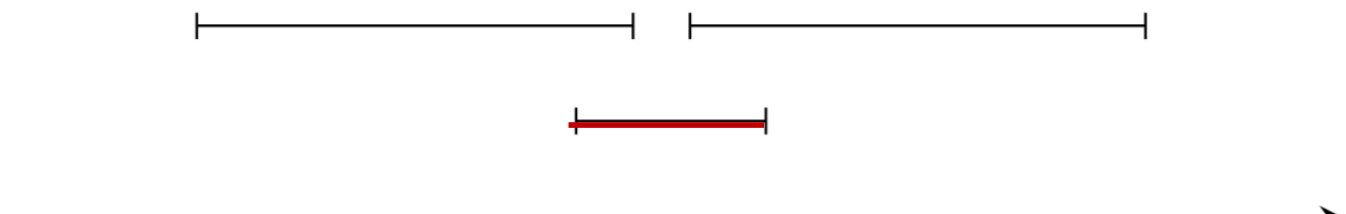


Figure: counterexample for shortest interval

Does Fewest Conflicts Choice Work?

- No

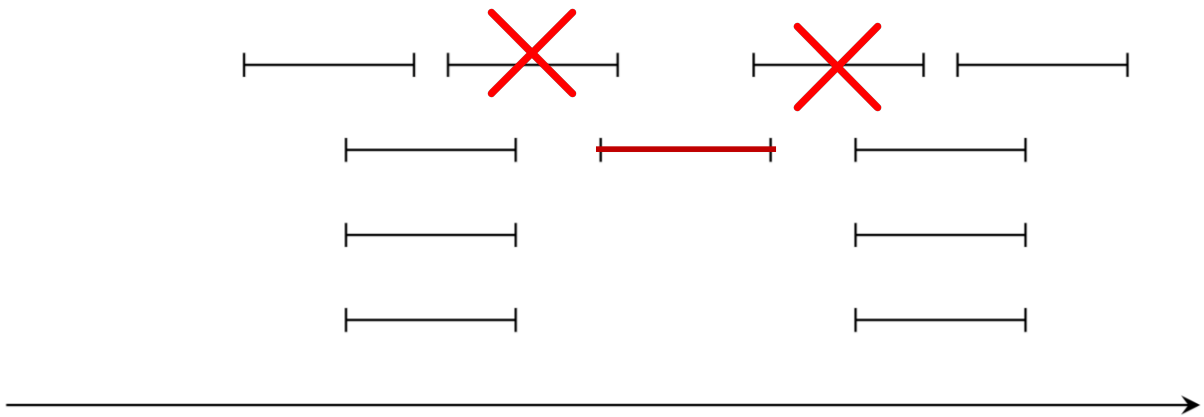


Figure: counterexample for fewest conflicts

Does Fewest Conflicts Choice Work?

- No

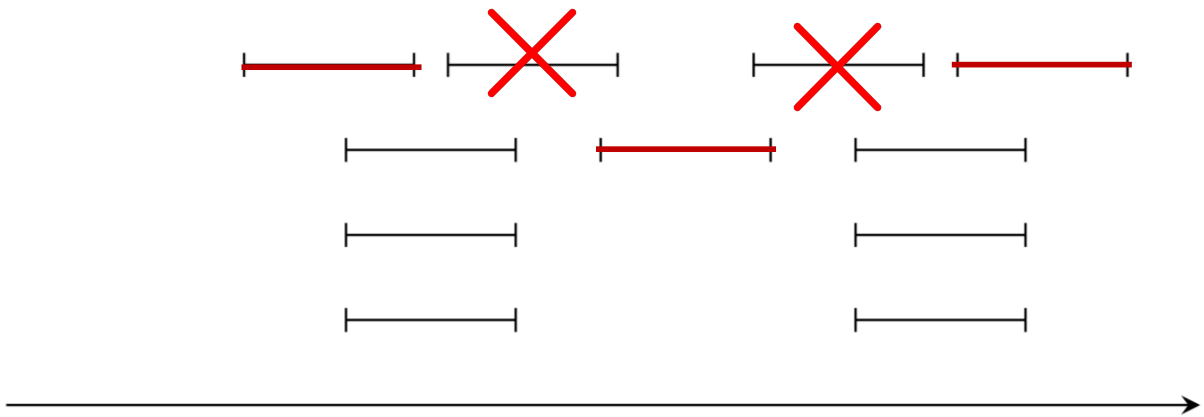


Figure: counterexample for fewest conflicts

Remaining Choice: Earliest Finish Time

- Fortunately, this turns out to work

```
GreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R be the set of all jobs;
2    Initialize X = 0    // X will store all scheduled jobs
3    while R is not empty
4        choose i ∈ R with smallest f_i;
5        add i to X
6        remove from R all jobs that overlap with i
7    return X
```

- Need to sort jobs in R according to f_i every time
- Better idea: sort all jobs at the very beginning

Remaining Choice: Earliest Finish Time

- Fortunately, this turns out to work

```
GreedyAlgo({s_i, f_i}_{i=1}^n)
1    Let R = set of all jobs sorted increasingly on f_i;
2    Initialize X = 0    // X will store all scheduled jobs
3    while R is not empty
4        choose i ∈ R with smallest f_i;
5        add i to X
6        remove from R all jobs that overlap with i
7    return X
```

- Need to sort jobs in R according to f_i every time
- Better idea: sort all jobs at the very beginning

Running Time Analysis

$O(n \log n)$

GreedyAlgo$(\{s_i, f_i\}_{i=1}^n)$

1 Let R = set of all jobs sorted increasingly on f_i;

2 Initialize $X = 0$ // X will store all scheduled jobs

3 **while** R is not empty

4 choose $i \in R$ with smallest f_i; → $O(1)$

5 add i to X

6 remove from R all jobs that overlap with i

7 **return** X

Naïve implementation $O(n)$:
check jobs one by one

In total: $O(n^2)$

Running Time Analysis

GreedyAlgo($\{s_i, f_i\}_{i=1}^n$)

1 Let R = set of all jobs sorted increasingly on f_i;
2 Initialize $X = 0$ // X will store all scheduled jobs
3 **while** R is not empty
4 choose $i \in R$ with smallest f_i;
5 add i to X
6 remove from R all jobs that overlap with i
7 **return** X

- Step 6 has a faster implementation where each job is visited once
  - Which job j would overlap with our choice i?
  - Exactly those with $s_j < f_i$ (since our choice implies $f_j \geq f_i$)
  - Can have another job order sorted according to s_j at the beginning; Step 6 simply removes all jobs before our chosen i in that order
- Running time under this implementation: $O(n \log n)$

Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- First: feasibility – does the algorithm output (at least) a valid solution?
  - Yes, by definition, as we removed all jobs conflicting with current schedules

Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- Second: optimality – is the output the best possible?

Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$

Prove $O = X$? → Not likely

Correctness Analysis

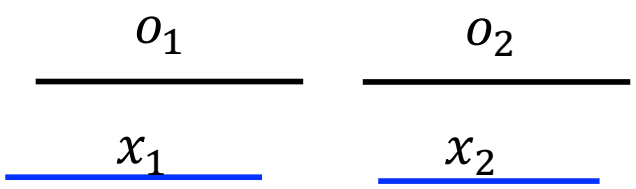
Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$ (note $k \leq m$)
  - Want to prove $k = m$, i.e., our output is equally good

Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

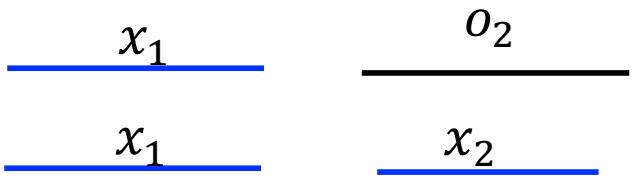
- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$ (note $k \leq m$)
  - Want to prove $k = m$, i.e., our output is equally good
  - Claim: for $i = 1, \cdots, k$, replacing o_i in optimal O by x_i maintain optimality and feasibility
    - Induction: when $i = 1$, this is true as x_1 finishes before o_1 and thus not conflicts with o_2



Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

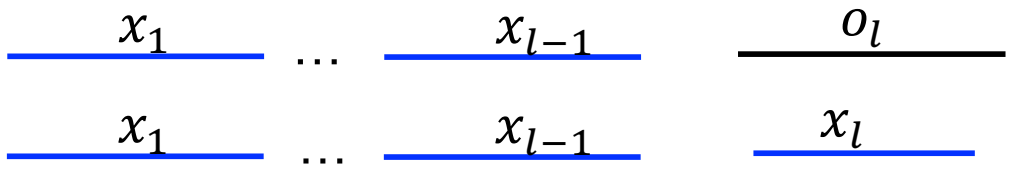
- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$ (note $k \leq m$)
  - Want to prove $k = m$, i.e., our output is equally good
  - Claim: for $i = 1, \cdots, k$, replacing o_i in optimal O by x_i maintain optimality and feasibility
    - Induction: when $i = 1$, this is true as x_1 finishes before o_1 and thus not conflicts with o_2



Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$ (note $k \leq m$)
  - Want to prove $k = m$, i.e., our output is equally good
  - Claim: for $i = 1, \cdots, k$, replacing o_i in optimal O by x_i maintain optimality and feasibility
    - Induction: when $i = 1$, this is true as x_1 finishes before o_1 and thus not conflicts with o_2
    - After replacement for all $i \leq l - 1$, can also do the replacement for $i = l$ because x_l has earliest finish time among all remaining non-conflicting jobs



Correctness Analysis

Theorem: The Greedy algorithm that picks the job with earliest finish time at each step is optimal.

- How to prove optimality?
  - Let the set of jobs achieves optimality be $O = \{o_1, \cdots, o_m\}$
  - Let our output set $X = \{x_1, \cdots, x_k\}$ (note $k \leq m$)
  - Want to prove $k = m$, i.e., our output is equally good
  - Claim: for $i = 1, \cdots, k$, replacing o_i in optimal O by x_i maintain optimality and feasibility
    - Induction: when $i = 1$, this is true as x_1 finishes before o_1 and thus not conflicts with o_2
    - After replacement for all $i \leq l - 1$, can also do the replacement for $i = l$ because x_l has earliest finish time among all remaining non-conflicting jobs
    - Since we can do this for any $i = 1, \cdots, k$, at the end the optimal solution becomes the same as X
    - So $k = m$

More About Interval Scheduling

- Question 1: the greedy algorithm of picking the shortest interval instead picks at least half the optimal number of jobs.
- Question 2: what if each job i has a reward r_i, you want to schedule jobs to maximize total rewards, without conflicts
  - The case we studied is $r_i = 1$ for all i
  - This will be in your HW 2
- Question 3: what if you do not know the jobs in advance; instead, they arrive in an online fashion
  - An active research area

Next Lectures: Graph Algorithms

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu