5112

8-24

Course Website

Interval Scheduling

Today

Finish up Gale-Shapley

Greedy Algorithms

- Interval Scheduling

Gale-Shapley Algorithm (1962)

while (∃ an unmatched university U):
  $C \leftarrow$ next candidate
  if (C is unmatched or prefers U):
    match C to U (and unmatch C if necessary)

For each university, maintain a list of candidates in order of preference. By next, mean next in this list.

How many times do we go through the loop?
Each university can make an offer to each candidate at most once.
⇒ At most $n \cdot n = n^2$, because there are n universities and n candidates.

Can we run out of candidates to make offers to?
Invariant: once a candidate receives an offer, they will have an offer for the rest of the algorithm.
If we reach the end of a candidate list, all candidates must have offers.
n candidates have offers ⟶ n universities have offers

What is the cost of each loop iteration?
- Need to find candidate preferences efficiently

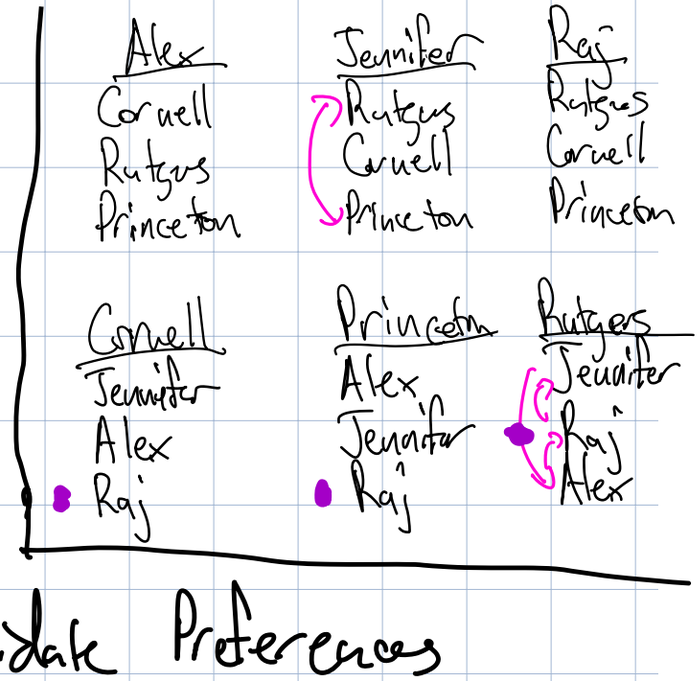
Gale-Shapley Algorithm

while (∃ an unmatched university U):
  C ← next candidate
  if (C is unmatched or prefers U):
    match C to U (and unmatch C if necessary)

For each university, maintain a list of candidates in order of preference. By next, mean next in this list.

What is the cost of each loop iteration?

- Need to find candidate preferences efficiently



Candidate Preferences

	Cornell	Princeton	Rutgers
Alex	1	3	2
Jennifer	2	3	1
Raj	2	3	1

Gale-Shapley Algorithm

while (∃ an unmatched university u):
  $c \leftarrow$ next candidate
  if (c is unmatched or prefers u):
    match c to u (and unmatch c if necessary)

For each university, maintain a list of candidates in order of preference. By next, mean next in this list.

Is the output stable?

Observation:
Candidates' matches only improve

Proof the output is stable:

Let c, u' be an unmatched pair, and u be c's match and c' be u''s match.
matches: c, u and c', u'.

Suppose u prefers c' to c.
Then u made an offer to c'.

not an unstable pair →

Case 1: c' rejected offer
⇒ c' already had a better
⇒ c' prefers u' to u. By obs.

Case 2: c' accepted u
But then at some point c' accepts u'
⇒ c' prefers u' to u By obs. ☐

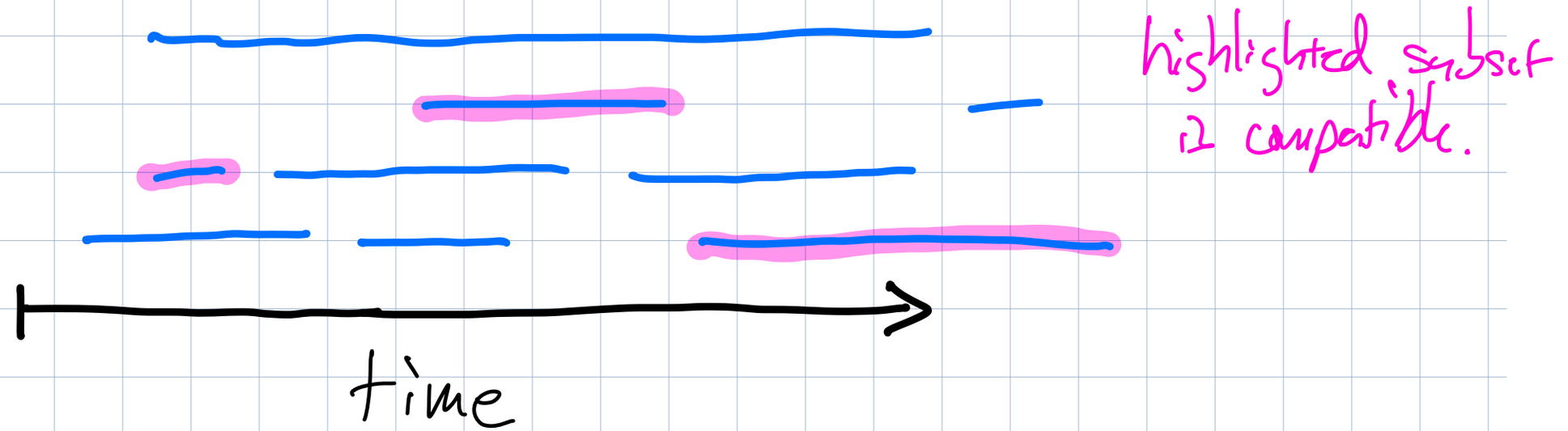
Greedy Algorithms

A greedy algorithm proceeds in steps, making a locally optimal decision at each step, without ever looking at the future.

Usually easy to come up with greedy algorithms but hard to show they solve the problem.
Whether they work depends on what you optimize.

Interval Scheduling

Input: $J_1, J_2, \ldots, J_n$ $\quad J_i = (s_i, f_i)$



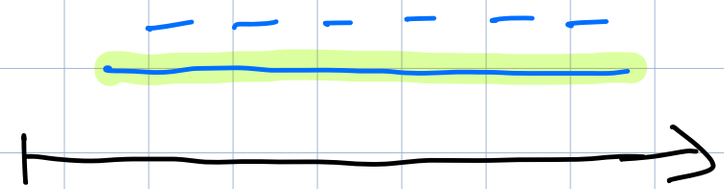
A subset of intervals is called <u>compatible</u> if no 2 overlap.

Goal: Find the largest compatible subset. (most jobs)

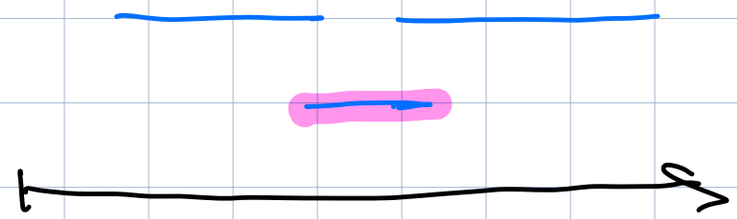
Interval Scheduling

Greedy Algorithms

1. Earliest Start Time (FCFS)
   (not correct)

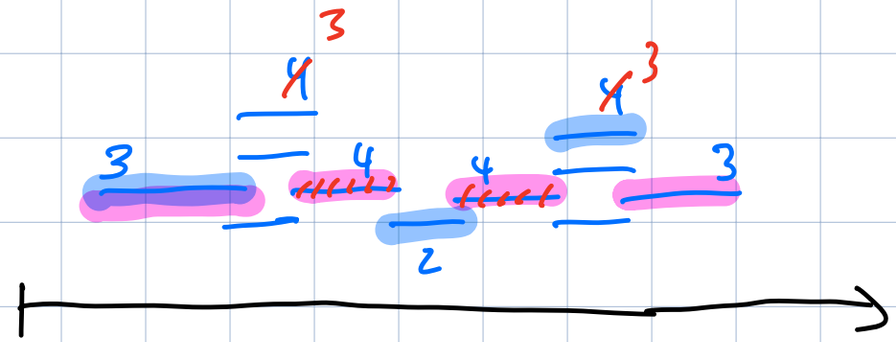


2. Shortest Job
   (not correct)

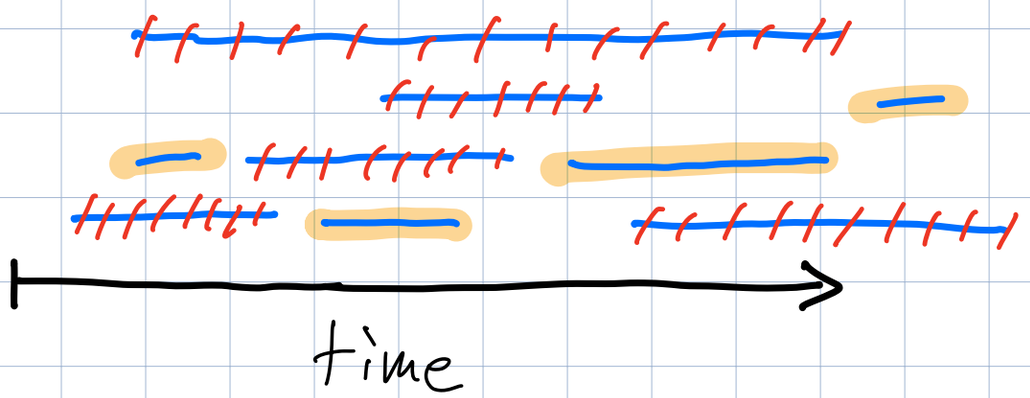


3. Exhaustive Search
   (correct, exponential time)

4. Fewest Conflicts (not correct)



5. Earliest Finish Time (looks promising)



Intuition: Choose the job that frees the resource as soon as possible.

Running Time of EFT:

Sort all the jobs by finish time. $O(n \log n)$

Start w/ earliest finishing job J_1. (Rename jobs in EFT order)

Then for J_2, if $(s_2 < f_1)$ then discard.

Repeat until we don't discard, then replace J_1

Running Time of EFT:

Sort all the jobs by finish time. $O(n \log n)$

$O(n)$

```
S = {J_1}
L ← J_1
i = 2
while (J_i):
    if (s_i ≥ f(L)):        // finish time of L
        S.append {J_i}      // J_i doesn't conflict w/ L
        L ← J_i
    i++
```

Iterate through each job once.

Total Runtime $O(n \log n)$

Does it produce a largest compatible subset?

Lemma: Let S be the output of EFT.
Let T be any compatible subset.
Order all jobs in S and T by EFT.

$K_1, \ldots, K_{|S|}$ $\quad$ $L_1, \ldots, L_{|T|}$

Then $f(K_i) \leq f(L_i)$ for all i.

Then $|S| \geq |T|$ for any T.