The Design and Analysis of Algorithms

Lecture 6 Greedy Algorithms I

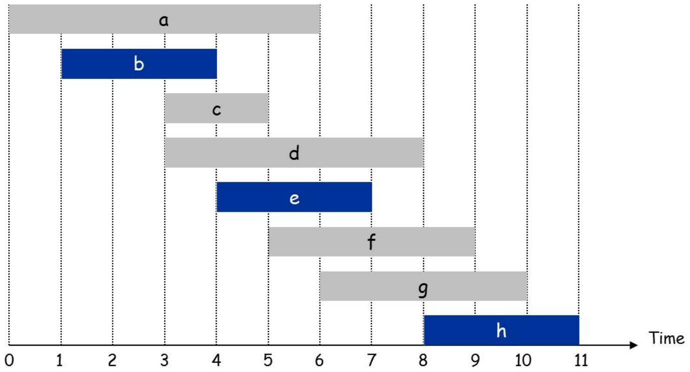
Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Interval Scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs *compatible* if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy Algorithms

- *Greedy template*. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.

(a) [Earliest start time] Consider jobs in ascending order of s_j.

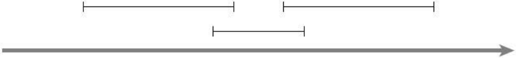
(b) [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

(c) [Fewest conflicts] For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.

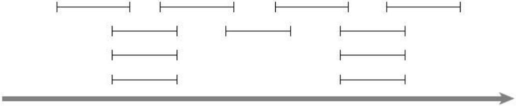
(d) [Earliest finish time] Consider jobs in ascending order of f_j.

Counterexamples for Greedy (a), (b) and (c)

(a)



(b)



(c)

Earliest-Finish-Time-First Algorithm

$EARLIEST - FINISH - TIME - FIRST(n, s_1, \cdots, s_n, f_1, \cdots, f_n)$

SORT jobs by finish time so that $f_1 \le f_2 \le \cdots \le f_n$.
$A \leftarrow \emptyset$
for $j = 1$ to n **do**
    if job j is compatible with A **then**
        $A \leftarrow A \cup \{j\}$.
    end if
end for
return A.

- *Proposition*. Can implement earliest-finish-time first in $O(n \log n)$ time.

  Keep track of job j^* that was added last to A.

  Job k is compatible with A iff $s_k \ge f_{j^*}$.

  Sorting by finish time takes $O(n \log n)$ time.

Analysis of Earliest-Finish-Time-First Algorithm

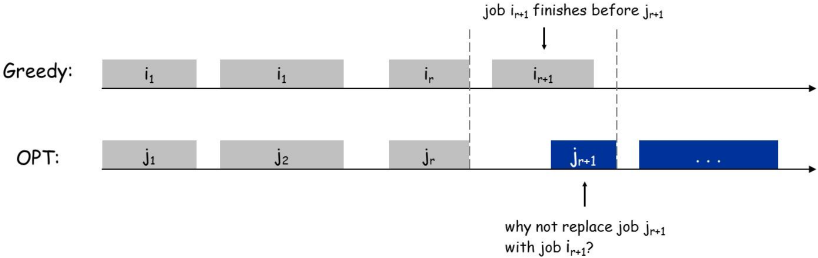
Theorem 1

The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

Assume greedy is not optimal. Let $i_1, i_2, \cdots, i_k$ denote set of jobs selected by greedy.

Let $j_1, j_2, \cdots, j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i2 = j2, \cdots, i_r = j_r$ for the largest possible value of r.



Scheduling to Minimize Lateness

- *Minimizing lateness problem*.

  Single resource processes one job at a time.

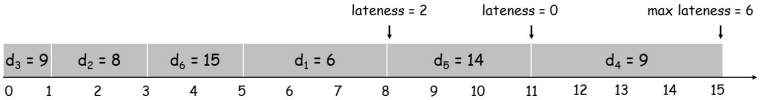
  Job j requires t_j units of processing time and is due at time d_j.

  If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.

  Lateness: $l_j = \max\{0, f_j - d_j\}$.

Goal: Schedule all jobs to minimize *maximum* lateness $L = \max_j l_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithms

- *Greedy template*. Schedule jobs according to some natural order.

  [Shortest processing time first] Schedule jobs in ascending order of processing time t_j.

  [Smallest slack] Schedule jobs in ascending order of slack $d_j - t_j$.

  [Earliest deadline first] Schedule jobs in ascending order of deadline d_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

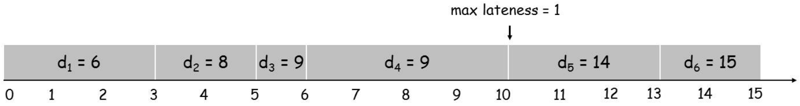
	1	2
t_j	1	10
d_j	2	10

counterexample

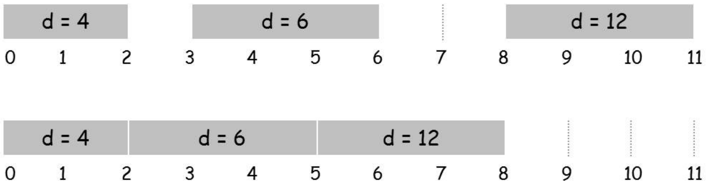
Minimizing Lateness: Earliest Deadline First

EARLIEST – DEADLINE – FIRST$(n, t_1, \cdots, t_n, d_1, \cdots, d_n)$

SORT jobs so that $d_1 \leq d_2 \leq \cdots \leq d_n$.
$t \leftarrow 0$
for $j = 1$ to n **do**
    Assign job j to interval $[t, t + t_j]$.
    $s_j \leftarrow t$; $f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$
end for
return Intervals $[s_1, f_1], [s_2, f_2], \cdots, [s_n, f_n]$.



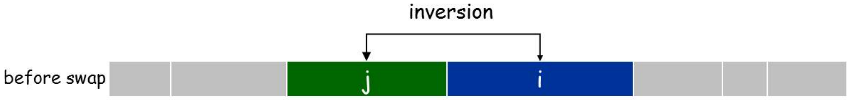
- *Observation 1*. There exists an optimal schedule with no idle time.
- *Observation 2*. The earliest-deadline-first schedule has no idle time.



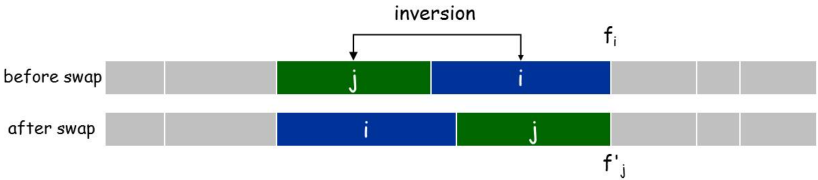
Minimizing Lateness: Inversions

Def. Given a schedule S, an *inversion* is a pair of jobs i and j such that: $d_i < d_j$ but j scheduled before i.

- *Observation 3*. The earliest-deadline-first schedule has no inversions.
- *Observation 4*. If a schedule (with no idle time) has an inversion, it has a pair of inverted jobs scheduled consecutively.



Minimizing Lateness: Inversions



Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let l be the lateness before the swap, and let l' be it afterwards.

$l'_k = l_k$ for all $k \neq i, j$.

$l'_i \leq l_i$.

$l'_j = f'_j - d_j = f_i - d_j \leq f_i - d_i \leq l_i$. □

Analysis of Greedy Algorithm

Theorem 2

The earliest-deadline-first schedule S is optimal.

Pf. [by contradiction] Define S^* to be an optimal schedule that has the fewest number of inversions.

Can assume S^* has no idle time.

If S^* has no inversions, then $S = S^*$.

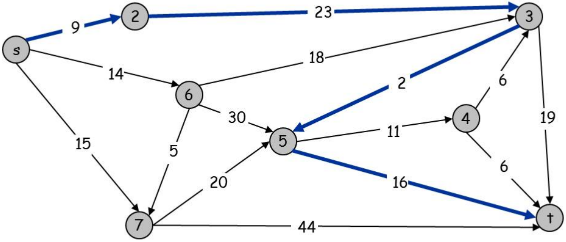
If S^* has an inversion, let i – j be an adjacent inversion.

Swapping i and j: Does not increase the max lateness, but strictly decreases the number of inversions.

This contradicts definition of S^*. □

Shortest-Paths Problem

- *Problem.* Given a digraph $G = (V, E)$, edge lengths $l_e \geq 0$, source $s \in V$, and destination $t \in V$, find the shortest directed path from s to t.



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Dijkstra's Algorithm

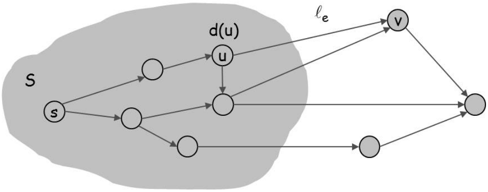
- Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

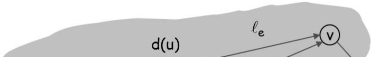
  Initialize $S = \{s\}$, $d(s) = 0$.

  Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e=(u,v):u \in S} d(u) + l_e,$$

  add v to S, and set $d(v) = \pi(v)$.





Dijkstra's Algorithm: Proof of Correctness

- *Invariant*. For each node $u \in S$, $d(u)$ is the length of the shortest $s \to u$ path.

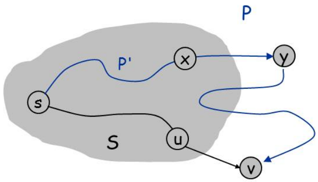
Pf. [by induction on $|S|$]

Base case: $|S| = 1$ is easy since $S = \{s\}$ and $d(s) = 0$.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

Let v be next node added to S, and let (u, v) be the final edge.

The shortest $s \to u$ path plus (u, v) is an $s \to v$ path of length $\pi(v)$.



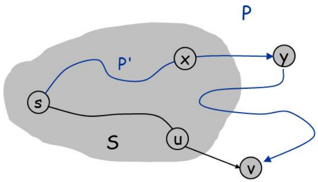
Dijkstra's Algorithm: Proof of Correctness–Con't

Consider any $s \to v$ path P. We show that it is no shorter than $\pi(v)$.

Let (x, y) be the first edge in P that leaves S, and let P' be the subpath to x.

P is already too long as soon as it reaches y.

$$l(P) \geq l(P') + l(x, y) \geq d(x) + l(x, y) \geq \pi(y) \geq \pi(v). \square$$



Dijkstra's Algorithm

Dijkstra' sAlgorithm(G, l)

```
Let S be the set of explored nodes.
For each u ∈ S, we store a distance d(u).
Initially S ← {s} and d(s) ← 0.
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for
    which d′(v) = min_{e=(u,v):u∈S} d(u) + l_e is as small as possible.
    Add v to S and define d(v) ← d′(v)
end while
return S.
```

Theorem 3

Dijkstra's algorithm can find the shortest path in $O(n^2)$ *time.*

Homework

- Read Chapter 4 of the textbook.
- Exercises 4 & 6 in Chapter 4.