



Design and Analysis of Algorithms

Greedy Algorithms

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com



Topics

- **Coin Changing**
- **Interval Scheduling and Partitioning**



Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest coins.

Ex. 34¢



Cashier's algorithm. At each iteration, and coin of the largest value that does not take us past the amount to be paid.

Ex. \$2.89





Cashier's Algorithm

At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Cashiers-Algorithm (x, c_1, c_2, \dots, c_n)

Sort n coin denominations so that $0 < c_1 < c_2 < \dots < c_n$

$S \leftarrow \emptyset$

while $x > 0$

$k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$

if no such k , **Return** “no solution”

else

$x \leftarrow x - c_k$

$S \leftarrow S \cup \{k\}$

Return S

Is Cashier's Algorithm optimal?



Properties of Optimal Solution

Property. Number of pennies ≤ 4 .

Pr. Replace 5 pennies with 1 nickel.

Property. Number of nickels ≤ 1 .

Property. Number of quarters ≤ 3 .

Property. Number of nickels + number of dimes ≤ 2 .

Pr.

- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.
- Recall: at most 1 nickel.



quarters
(25¢)



dimes
(10¢)



nickels
(5¢)



pennies
(1¢)



Analysis of Cashier's Algorithm

Theorem. Cashier's algorithm is optimal for U.S. coins: 1, 5, 10, 25, 100.

Pf.

- Consider optimal way to change $c_k \leq x \leq c_{k+1}$: greedy takes coin k .
- We claim that any optimal solution must also take coin k .
 - if not, it needs enough coins of type c_1, \dots, c_{k-1} to add up to x .
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by Cashier's algorithms.



Cashier's Algorithm for Other Denominations

Is Cashier's algorithm optimal for any set of denominations?

No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Cashier's algorithm: $140\text{¢} = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1$.
- Optimal: $140\text{¢} = 70 + 70$.



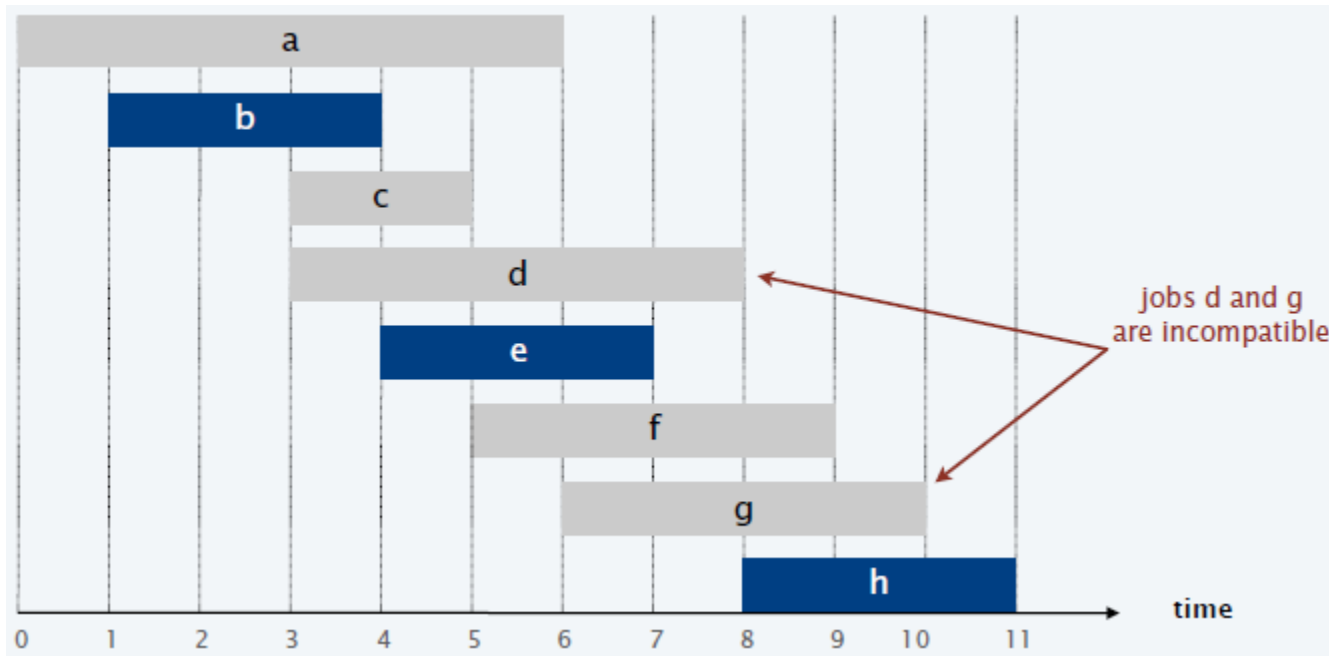
It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

- Cashier's algorithm: $15\text{¢} = 9 + ?$.
- Optimal: $15\text{¢} = 7 + 8$.



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.

- [Earliest start time] Consider jobs in ascending order of s_j .
- [Earliest finish time] Consider jobs in ascending order of f_j .
- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each job j , count the number of conflicting jobs c_j . Schedule in ascending order of c_j .



Interval Scheduling: Greedy Algorithms

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

counterexample for earliest start time



counterexample for shortest interval



counterexample for fewest conflicts





Interval Scheduling: Earliest-Finish-Time-First Algorithm

Earliest-Finish-Time-First $(n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n)$

Sort jobs by finish time so that $f_1 \leq f_2 \leq \dots \leq f_n$

$A \leftarrow \emptyset$ (set of jobs selected)

for $j = 1$ to n

If job j is compatible with A

$A \leftarrow A \cup \{j\}$

Return A

The Earliest-Finish-Time-First algorithm is optimal.

Proposition. Can implement Earliest-Finish-Time-First in ? time.



Interval Scheduling: Earliest-Finish-Time-First Algorithm

Earliest-Finish-Time-First $(n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n)$

Sort jobs by finish time so that $f_1 \leq f_2 \leq \dots \leq f_n$

$A \leftarrow \emptyset$ (set of jobs selected)

for $j = 1$ to n

If job j is compatible with A

$A \leftarrow A \cup \{j\}$

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 j is compatible with A iff $s_j \geq f_{j^*}$.
- Sorting by finish time takes $O(n \log n)$ time.

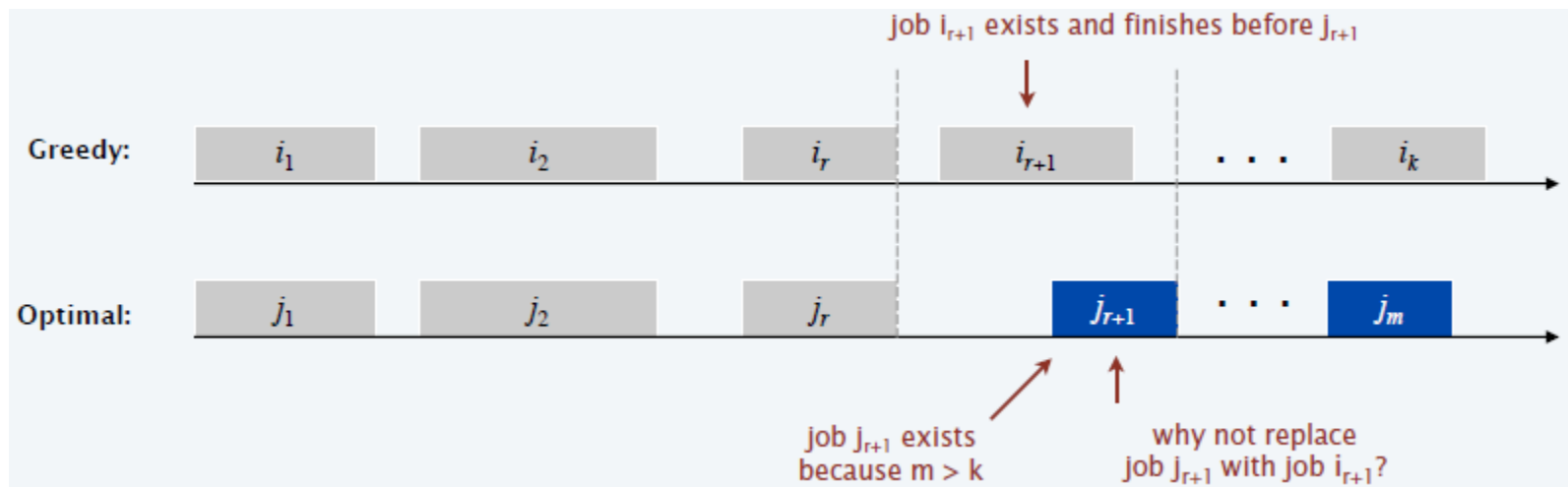


Interval Scheduling: Analysis of Earliest-Finish-Time-First Algorithm

Theorem. The Earliest-Finish-Time-First algorithm is optimal.

Pf.

- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, \dots, i_k denote set of jobs selected by greedy.
- Let j_1, j_2, \dots, j_m denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r .



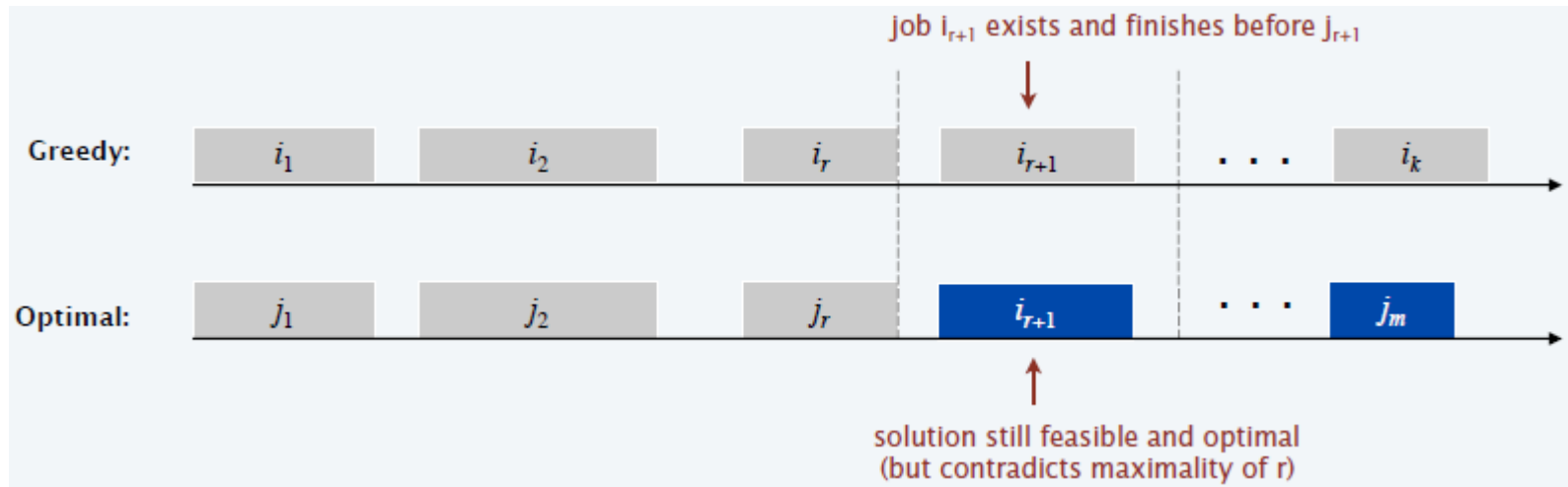


Interval Scheduling: Analysis of Earliest-Finish-Time-First Algorithm

Theorem. The Earliest-Finish-Time-First algorithm is optimal.

Pf.

- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, \dots, i_k denote set of jobs selected by greedy.
- Let j_1, j_2, \dots, j_m denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r .



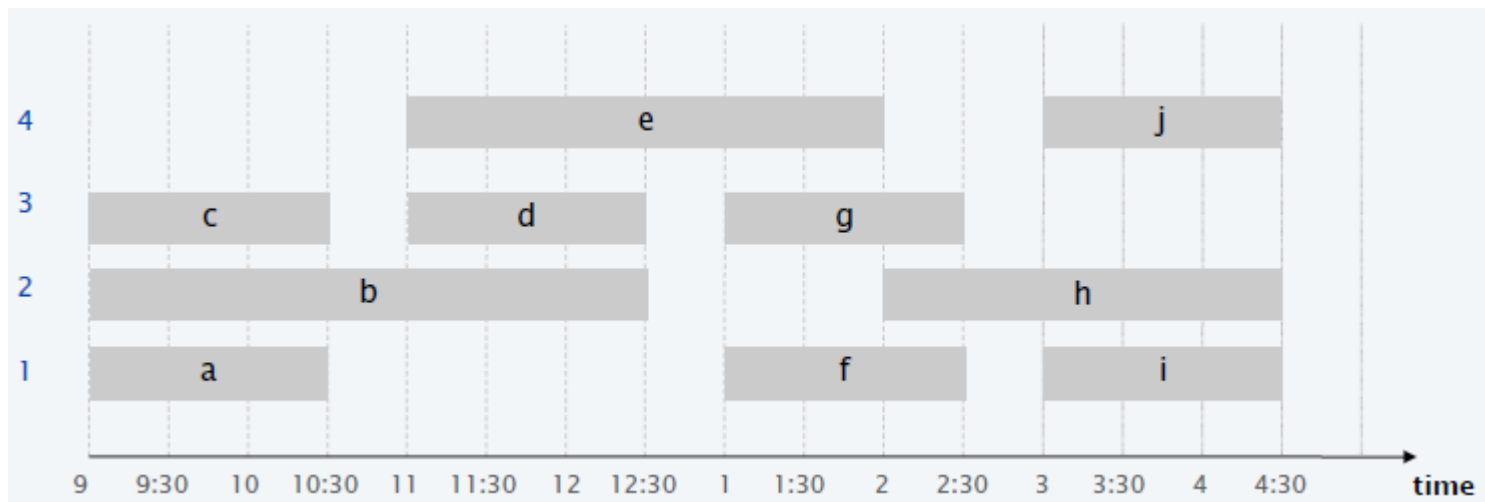


Interval Partitioning

Interval Partitioning.

- Lecture j starts at s_j and finishes at f_j .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.



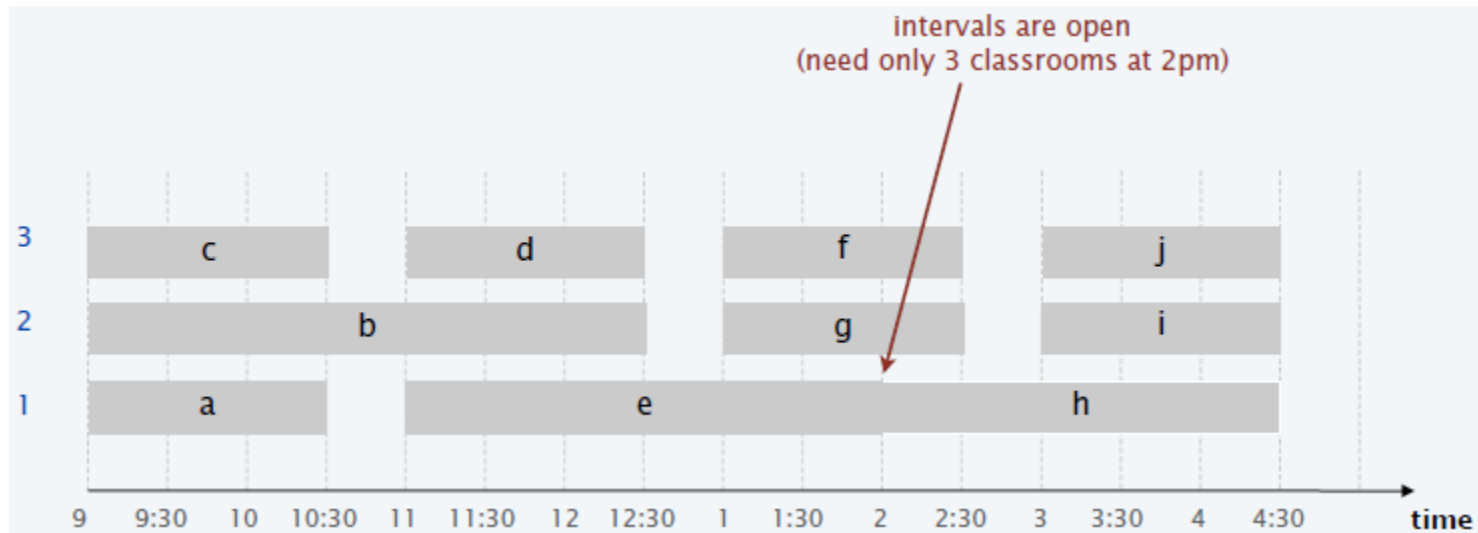


Interval Partitioning

Interval Partitioning.

- Lecture j starts at s_j and finishes at f_j .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.





Interval Partitioning: Greedy Algorithm

Greedy template. Consider lectures in some natural order. Assign each lecture to an available classroom; allocate a new classroom if none are available.

- [Earliest start time] Consider lectures in ascending order of s_j .
- [Earliest finish time] Consider lectures in ascending order of f_j .
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each lectures j , count the number of conflicting lectures c_j . Schedule in ascending order of c_j .



Interval Partitioning: Greedy Algorithm

Greedy template. Consider lectures in some natural order. Assign each lecture to an available classroom; allocate a new classroom if none are available.





Interval Partitioning: Earliest-Start-Time-First Algorithm

Earliest-Start-Time-First ($n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$)

Sort lectures by start time so that $s_1 \leq s_2 \leq \dots \leq s_n$

$d \leftarrow 0$ (the number of allocated classrooms)

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

if lecture j is compatible with some classroom

 Schedule lecture j in any such classroom k .

else

 Allocate a new classroom $d + 1$.

 Schedule lecture j in classroom $d + 1$.

$d \leftarrow d + 1$

Return schedule.



Interval Partitioning: Earliest-Start-Time-First Algorithm

Proposition. The Earliest-Start-Time-First algorithm can be implemented in $O(n \log n)$ time.

Pf. Store classrooms in a priority queue (key=finish time of its last lecture).

- To determine whether lecture j is compatible with some classroom, compare s_j to key of min classroom k in priority queue.
- To add lecture j to classroom k , increase key of classroom k to f_j .
- Total number of priority queue operation is $O(n)$.
- Sorting by start time takes $O(n \log n)$ time.

Remark. This implementation chooses a classroom k whose finish time of its last lecture is the earliest.



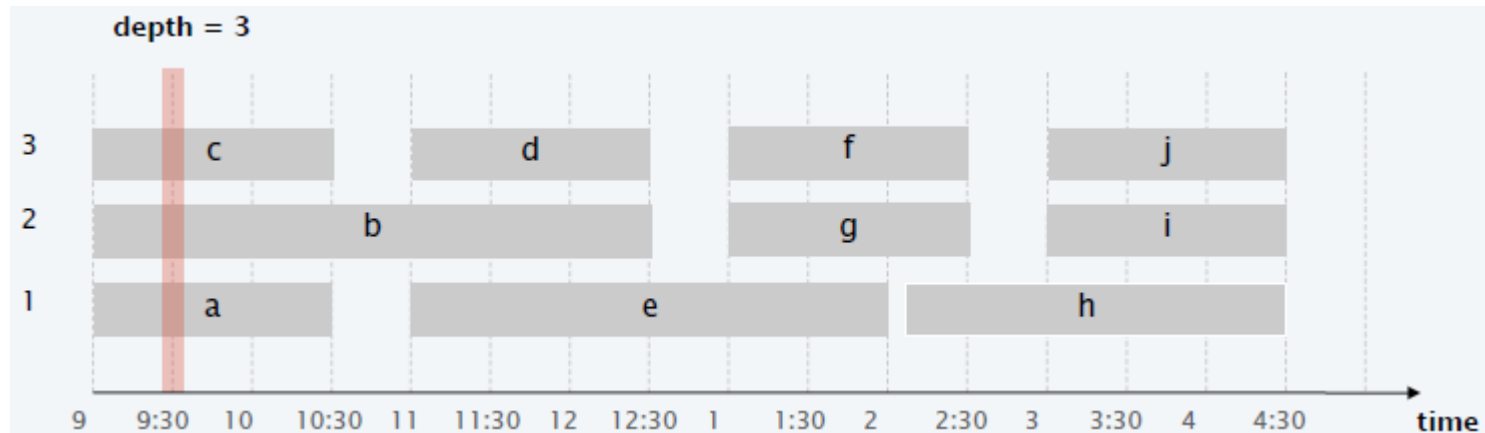
Interval Partitioning: Lower Bound on Optimal Solution

Def. The depth of a set of open intervals is the maximum number of intervals that contain any given time.

Key observation. Number of classrooms needed \geq depth.

Does minimum number of classrooms needed always equal depth?

Earliest-Start-Time-First algorithm finds a schedule whose number of classrooms equals the depth.





Interval Partitioning: Analysis of Earliest-Start-Time-First Algorithm

Observation. The Earliest-Start-Time-First algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-Start-Time-First algorithm is optimal.
Pf.

- Let d = number of classrooms that the algorithm allocates.
- Classroom d is opened because we needed to schedule a lecture, say j , that is incompatible with all $d - 1$ other classrooms.
- These d lectures each end after s_j .
- Since we sorted by start time, all these incompatibilities are caused by the lectures that start no later than s_j .
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- Key observation \Rightarrow all schedules use $\geq d$ classrooms.