COM2031 Advanced Algorithms

Topic 2: Greedy Algorithms

Steve Schneider

Greedy Algorithms

Greedy Algorithms

- build up a solution incrementally in small steps using some local decision rule,
- make a decision at each step just based on the solution so far
- Probably the most simple and straight forward way to design an algorithm.

Key feature:

- Only need to see the solution so far to make a decision about what to do next.
- The sequence of decisions about what to do next leads to an optimal solution.

The General Scenario:

- A collection of items, need to select some to optimize something
- Consider items in some order, consider each in turn, and select it if it is compatible with the previous selections

Greedy Algorithm Template

- Consider items in a particular order.
  - The initial step may involve a preprocessing step of **sorting** the items if they are not already sorted
- Take each item provided it is compatible with the ones already taken, otherwise discard it.
  - There will be some check on the item and the solution so far
- Continue until all items have been considered or a solution is reached.

The greedy algorithm is straightforward. But it is also necessary to reason why it gives the optimal solution.

The order in which items are considered is critical – considering the items in the wrong order may give a non-optimal solution

COIN CHANGE PROBLEM

Coin Change problem

Problem: Find the minimum number of coins to make a specific amount *n*

Coin denominations: 1, 2, 5, 10, 20, 50, 100, 200. We have as many of each coin as we need.

Greedy Algorithm:

- Sort the denominations largest to smallest:
  - 200, 100, 50, 20, 10, 5, 2, 1
- Keep taking the largest coin which can fit into the remaining amount
- Stop when you have the exact amount

Coin Change problem

Example: coins to make n = 582

Take 200:	382 remaining
Take 200:	182 remaining
Take 100:	82 remaining
Take 50:	32 remaining
Take 20:	12 remaining
Take 10:	2 remaining
Take 2:	0 remaining – **done!**

Solution: {200, 200, 100, 50, 20, 10, 2} : 7 coins.

Coin Change problem – wrong order

Example: coins to make n = 582

What if we considered coins from smallest to largest?

Take 1: 581 remaining

Take 1: 580 remaining

… etc

Solution: {1,1,…,1} : 582 coins.

Not optimal! The order is critical

COIN CHANGE PROBLEM

COIN CHANGE PROBLEM

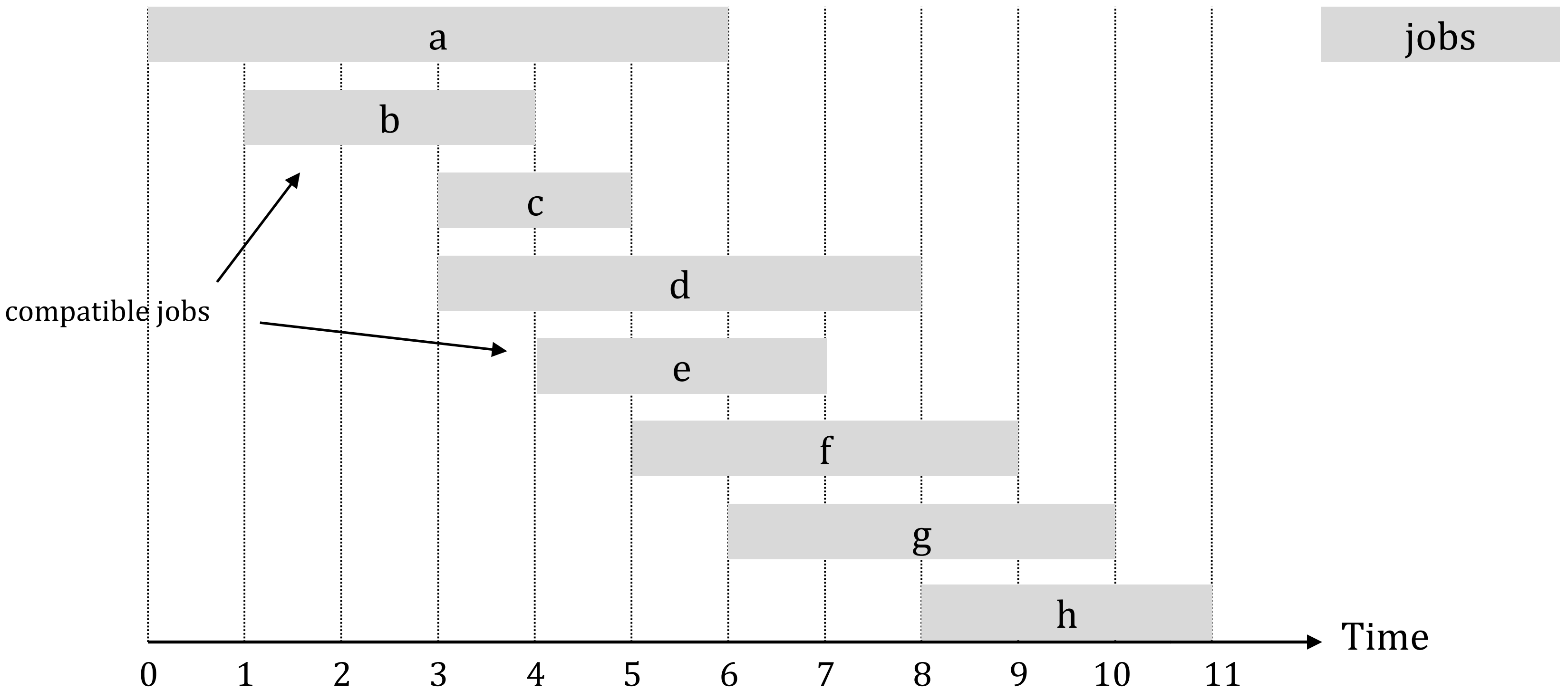
GREEDY ALGORITHM WORKS WHEN WE HAVE DEMONINATIONS {1,2,5,10, 20, 50, 100, 200}

INTERVAL SCHEDULING PROBLEM

Interval Scheduling Problem

Interval Scheduling Problem

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



Interval Scheduling: Greedy Approach

Greedy Algorithm Template

- Consider jobs in a particular order.
- Take each job provided it is compatible with the ones already taken.

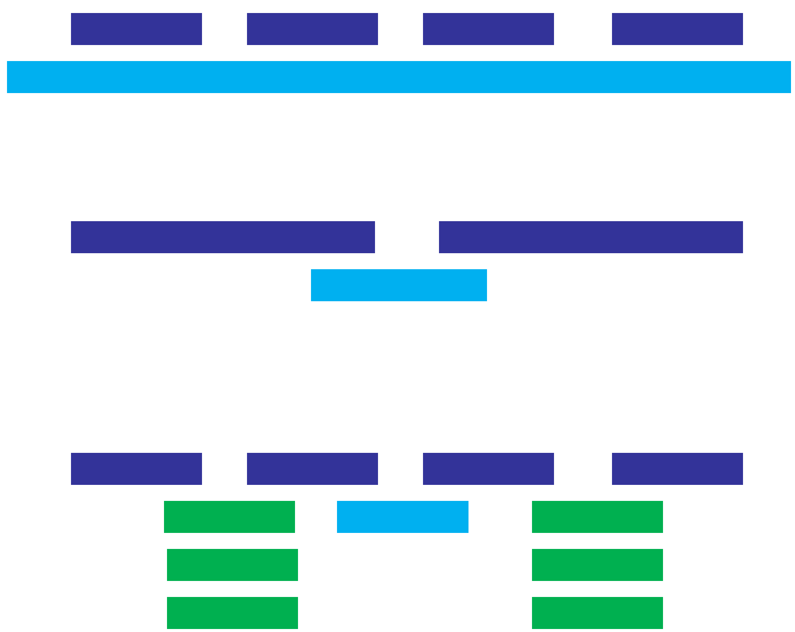
A priori, many ways to **order** jobs:

- Earliest start time — Consider jobs in ascending order of start time s_j
  (allows to use available resources quickly)
- Earliest finish time — Consider jobs in ascending order of finish time f_j
  (available resources will be released earlier)
- Shortest interval — Consider jobs in ascending order of interval length $f_j - s_j$
  (establishes priority for jobs that consume less time)
- Fewest conflicts — For each job, count the number of conflicting jobs c_j
  Schedule in ascending order of conflicts c_j

Interval Scheduling: Greedy Approach

Greedy Algorithm Template

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



earliest start time is not optimal

shortest intervals are not optimal

fewest conflicts are not optimal

optimal solution

selected job

Interval Scheduling: Greedy Algorithm

Greedy Algorithm for Interval Scheduling Problem

- Consider jobs in the order of earliest finish time.
- Take each job provided it is compatible with the ones already taken.

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

J ← φ (initialize set of selected jobs)
for j = 1 to n {
   if (job j compatible with J)
      J ← J ∪ {j}
}
return J
```

- Remember latest job j* added to J. New job j is compatible if $s_j \geq f_{j^*}$.
  That is only 1 comparison needed: O(1)
- **Running time:** O(n log n) time due to the sorting step.
- But how can we convince ourselves that this algorithm is doing the right thing?

Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

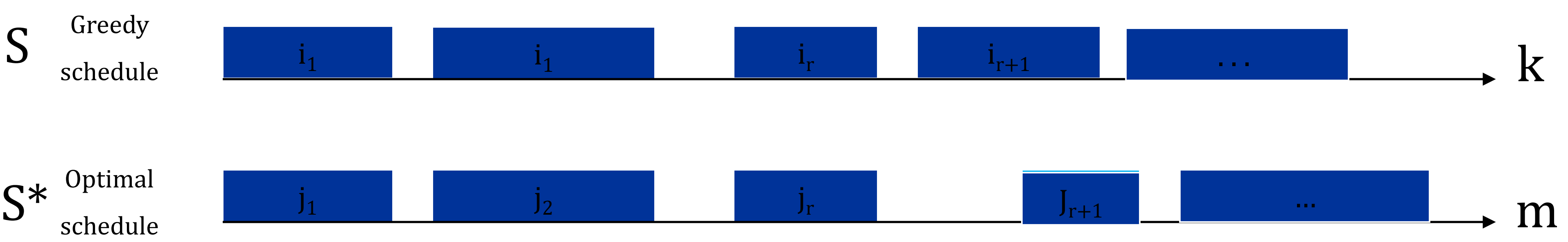
<u>Proof</u> Let S* be any optimal schedule and $j_1, j_2, \ldots j_m$ denote the jobs in it.

Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof</u> Let S* be any optimal schedule and $j_1, j_2, \ldots j_m$ denote the jobs in it.

- Let S be the output of our algorithm and $i_1, i_2, \ldots i_k$ denote jobs selected by the greedy algorithm. As S* is optimal $k <= m$.

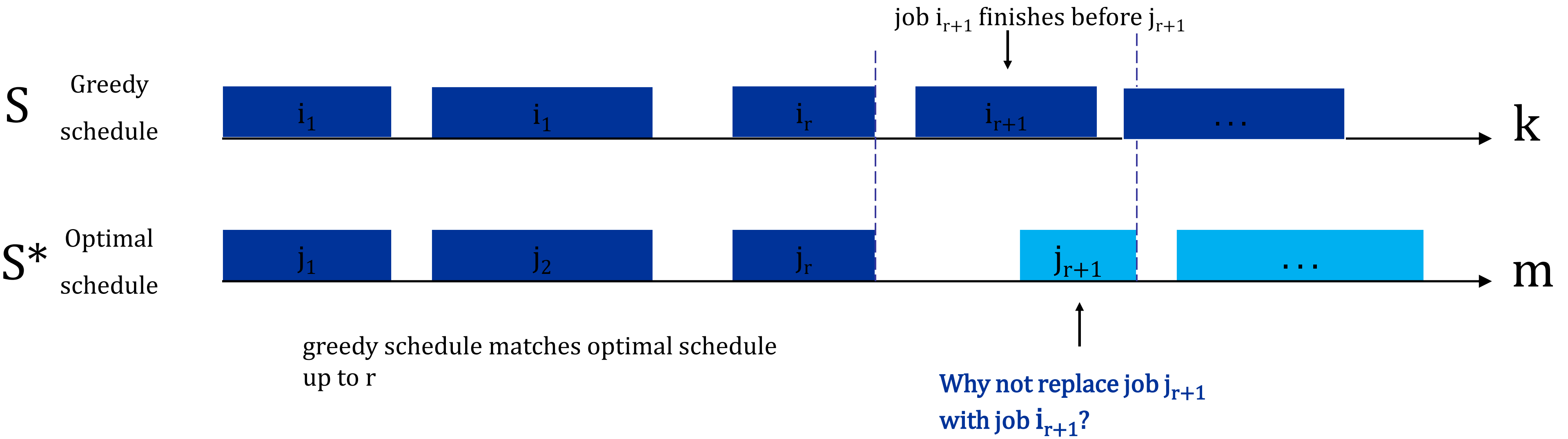


Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof</u> Let S* be any optimal schedule and $j_1, j_2, \ldots j_m$ denote the jobs in it.

- Let S be the output of our algorithm and $i_1, i_2, \ldots i_k$ denote jobs selected by the greedy algorithm. As S* is optimal $k <= m$.
- If S* and S agree up to index r, ie $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ then we can construct another optimal solution from S* by replacing j_{r+1} with i_{r+1}, see below, hence we have an optimal solution which agrees with S up to r+1.

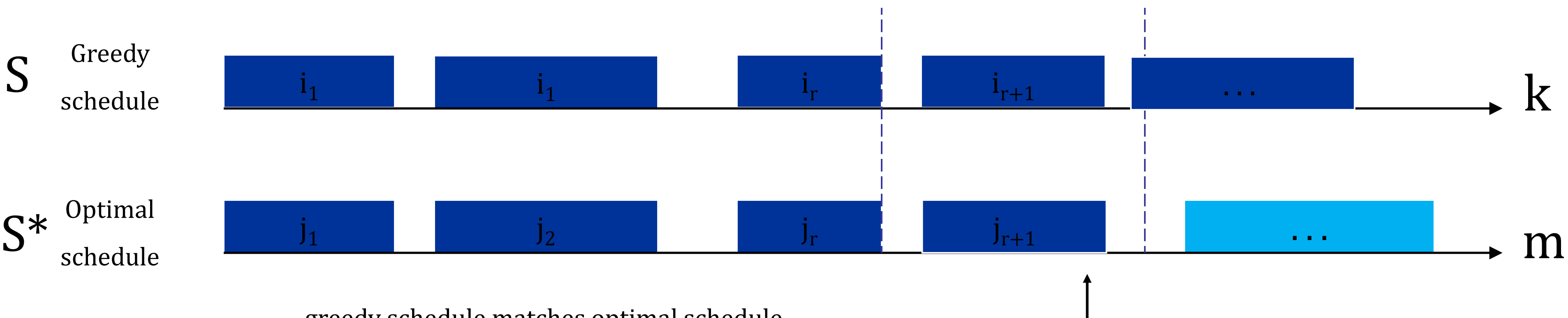


Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof</u> Let S* be any optimal schedule and $j_1, j_2, \ldots j_m$ denote the jobs in it.

- Let S be the output of our algorithm and $i_1, i_2, \ldots i_k$ denote jobs selected by the greedy algorithm. As S* is optimal $k <= m$.
- If S* and S agree up to index r, ie $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ then we can construct another optimal solution from S* by replacing j_{r+1} with i_{r+1}, see below, hence we have an optimal solution which agrees with S up to r+1.



greedy schedule matches optimal schedule up to r

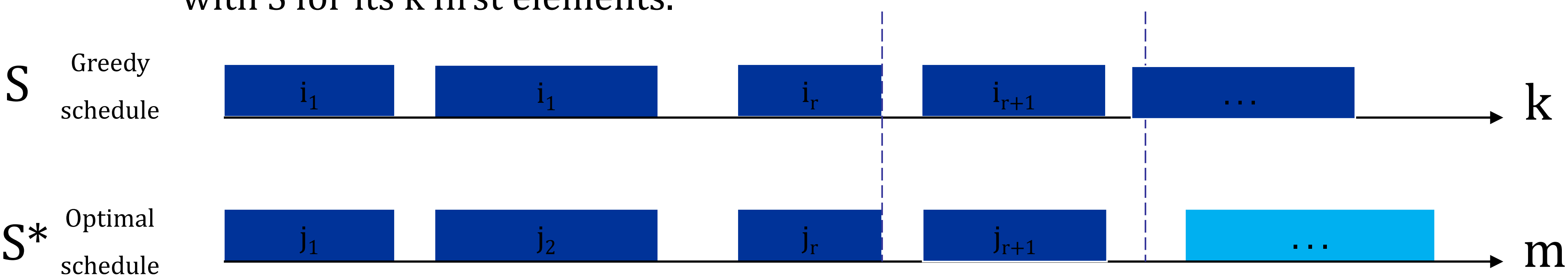
Why not replace job j_{r+1} with job i_{r+1}?

Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof</u> Let S^* be any optimal schedule and $j_1, j_2, \ldots j_m$ denote the jobs in it.

- Let S be the output of our algorithm and $i_1, i_2, \ldots i_k$ denote jobs selected by the greedy algorithm. As S^* is optimal $k <= m$.
- If S^* and S agree up to index r, ie $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ then we can construct another optimal solution from S^* by replacing j_{r+1} with i_{r+1}, see below, hence we have an optimal solution which agrees with S up to $r+1$.
- Repeating this procedure we end up with an optimal solution S^{**} that agrees with S for its k first elements.



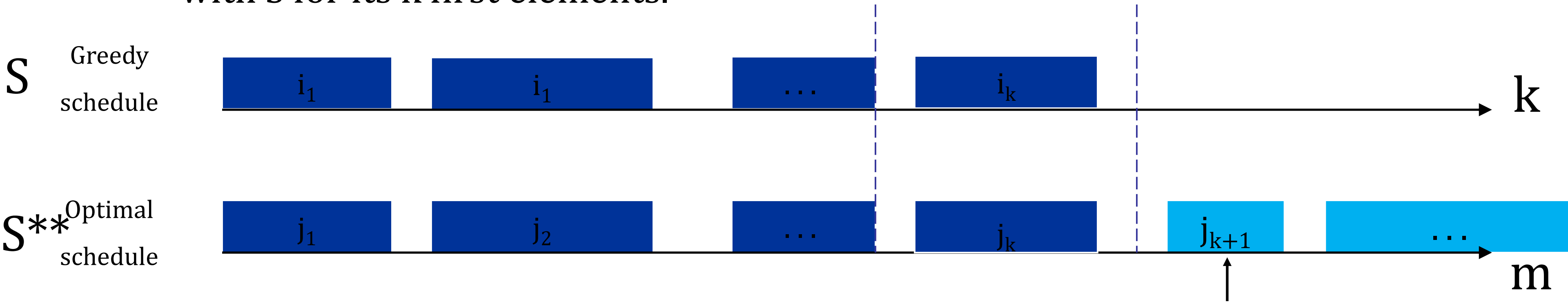
greedy schedule matches optimal schedule
up to r

Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof</u> Let S^* be any optimal schedule and $j_1, j_2, \dots j_m$ denote the jobs in it.

- Let S be the output of our algorithm and $i_1, i_2, \dots i_k$ denote jobs selected by the greedy algorithm. As S^* is optimal $k <= m$.
- If S^* and S agree up to index r, ie $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ then we can construct another optimal solution from S^* by replacing j_{r+1} with i_{r+1}, see below, hence we have an optimal solution which agrees with S up to r+1.
- Repeating this procedure we end up with an optimal solution S^{**} that agrees with S for its k first elements.

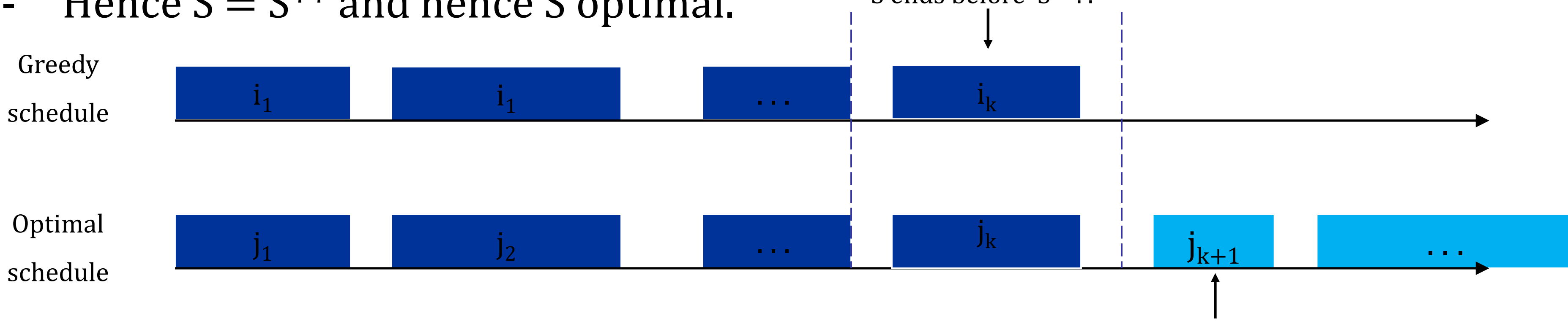


Interval Scheduling: Analysis

Theorem Greedy algorithm based on earliest finish time is optimal.

<u>Proof continued:</u> What remains to be shown is that m=k.

- We know already k <= m, but k < m cannot happen, because:
- In that case, there would be at least one job left, namely j_{k+1} that is compatible with i_k and has a later finish time.
- However the greedy algorithm runs until all such jobs are selected.
- Hence k<m cannot happen, and k=m.
- Hence S = S** and hence S optimal.



Greedy schedule matches optimal schedule in length.

But then there is at least one job compatible with i_k and with finish time later than i_k

INTERVAL SCHEDULING PROBLEM

GREEDY ALGORITHM SOLUTION WITH JOBS SORTED BY EARLIEST FINISH TIME

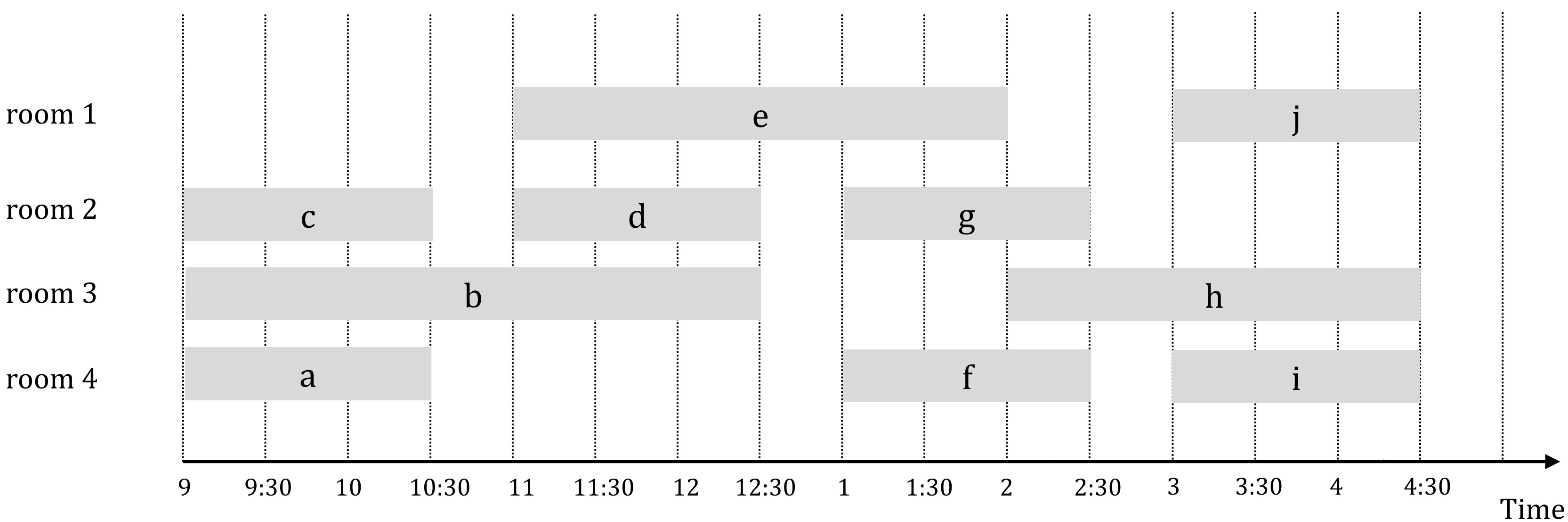
INTERVAL PARTITIONING

Interval Partitioning Problem

Interval Partitioning Problem

- 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 occur at the same time in the same room.

Example This schedule uses 4 classrooms to schedule 10 lectures.

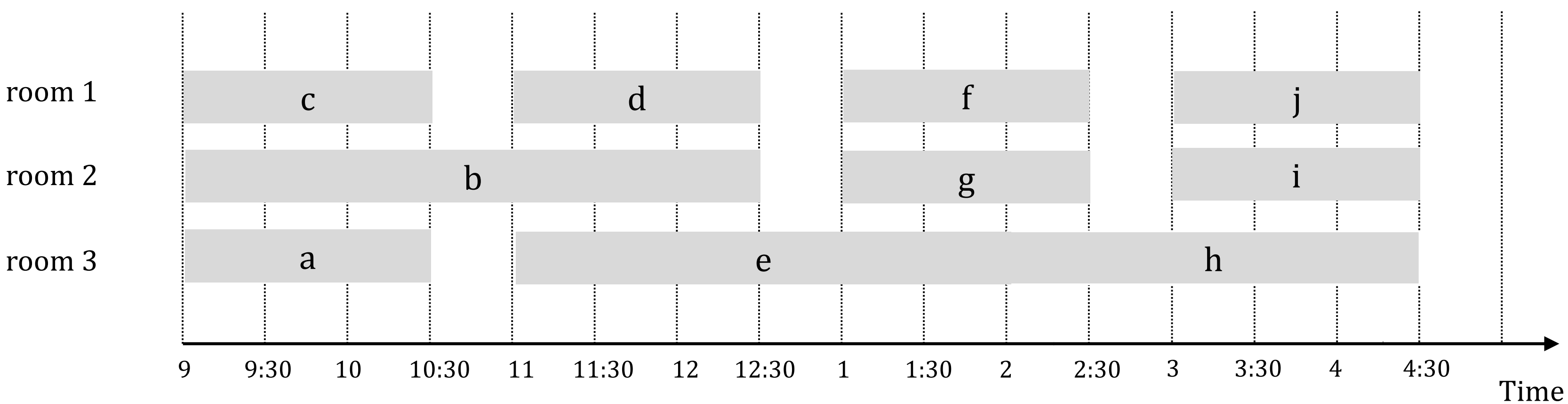


Interval Partitioning Problem

Interval Partitioning Problem

- 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 occur at the same time in the same room.

Example This schedule uses only 3 rooms for the same 10 lectures.



Interval Partitioning: Lower Bound on Optimal Solution

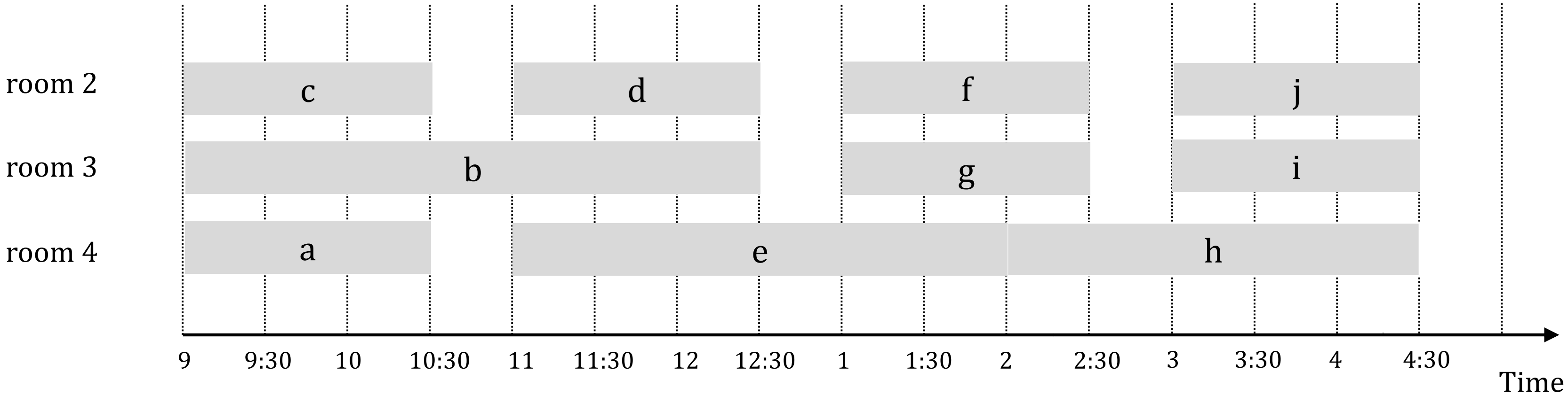
Definition The depth of a set of (open) intervals is the maximum number of intervals that contains any given time.

Observation Number of classrooms needed $\geq$ depth.

Example Depth of this schedule = 3 $\Rightarrow$ This schedule is optimal.

e.g. lectures a, b, c all contain 9:30

Does there always exist a schedule equal to depth of intervals?



Interval Partitioning: Greedy Algorithm

Greedy Algorithm for Interval Partitioning Problem

- Consider lectures in the order of earliest start time.
- Assign lecture to any compatible classroom.

```
Sort lectures by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0  (initialize the number of allocated classrooms)

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Running time O(n log n) due to sorting step.

Interval Partitioning: Analysis

Fact By construction, this greedy algorithm never schedules two incompatible (overlapping) lectures in the same classroom.

Theorem Greedy algorithm based on earliest start time is optimal.

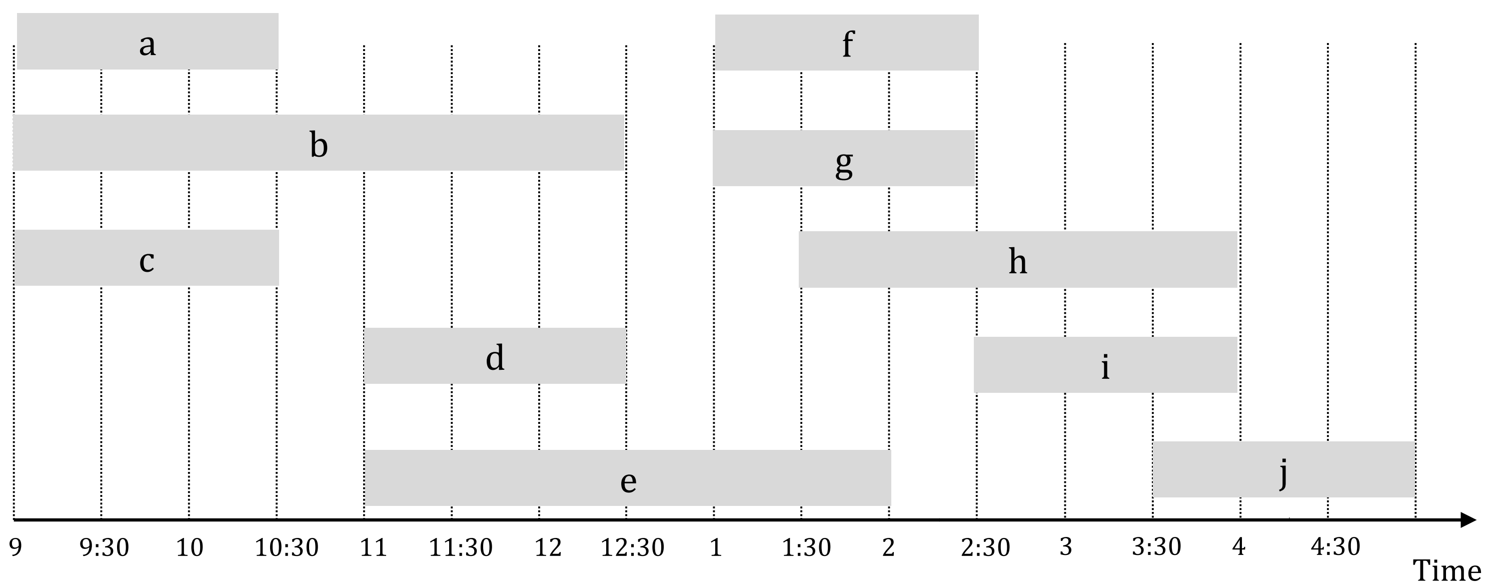
<u>Proof</u>

Let d denote the number of classrooms allocated:

- A new classroom d is opened only if a new lecture j that is incompatible with all existing d – 1 classrooms has to be scheduled.
- Sorting by earliest start time implies this new incompatibility is caused only by lectures that start no later than s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$ for some $\varepsilon > 0$.
- However Depth $\geq$ d $\Rightarrow$ this schedule uses classrooms no more than Depth.

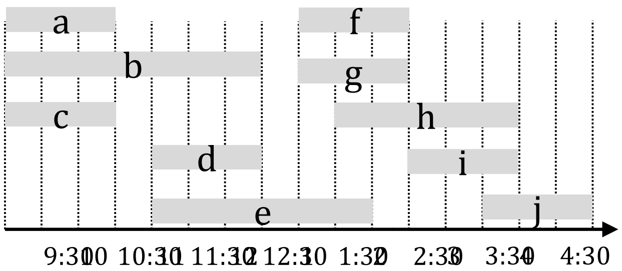
Interval Partitioning Problem

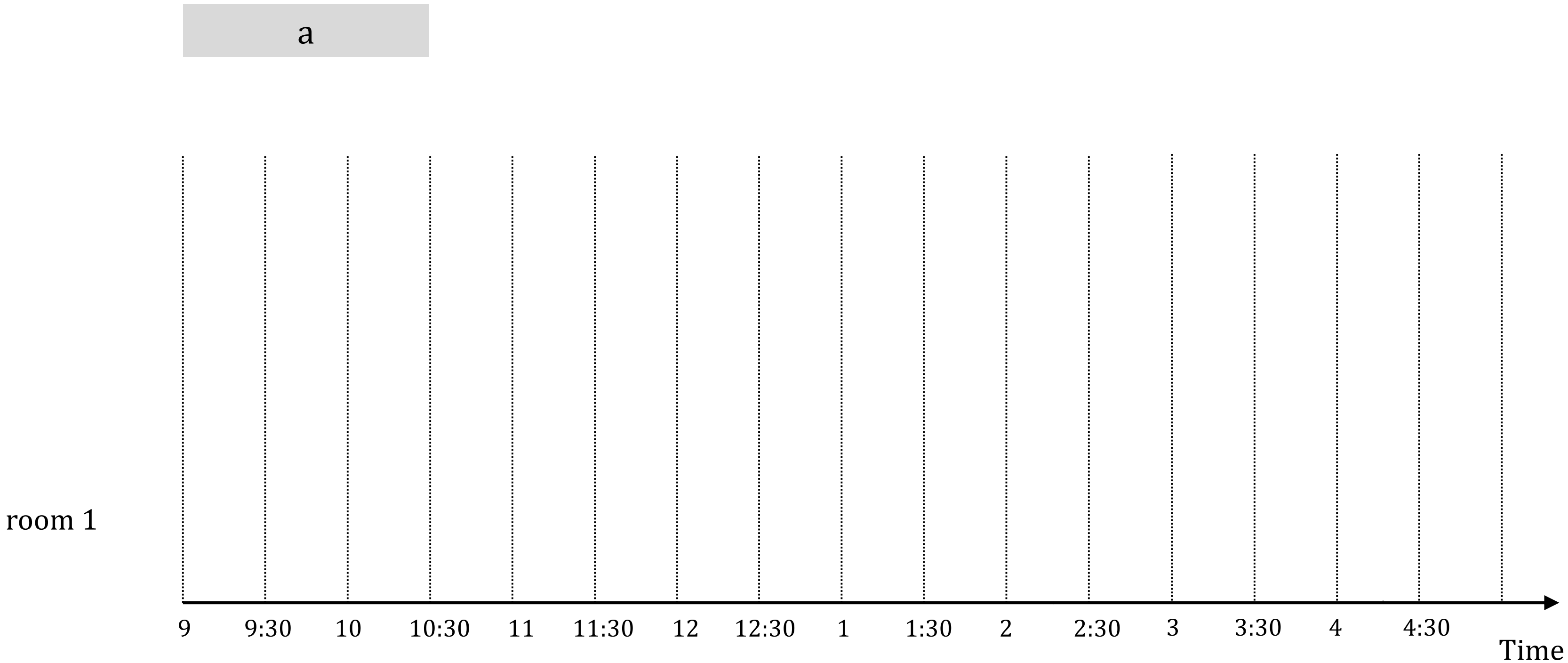
Example



Interval Partitioning Problem Example

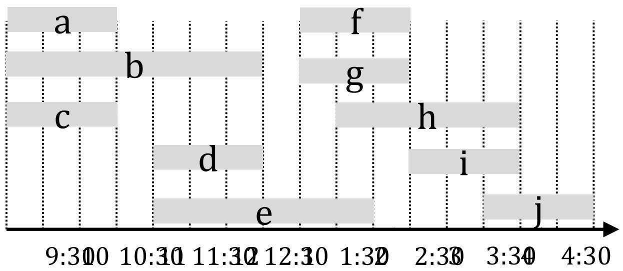
Example

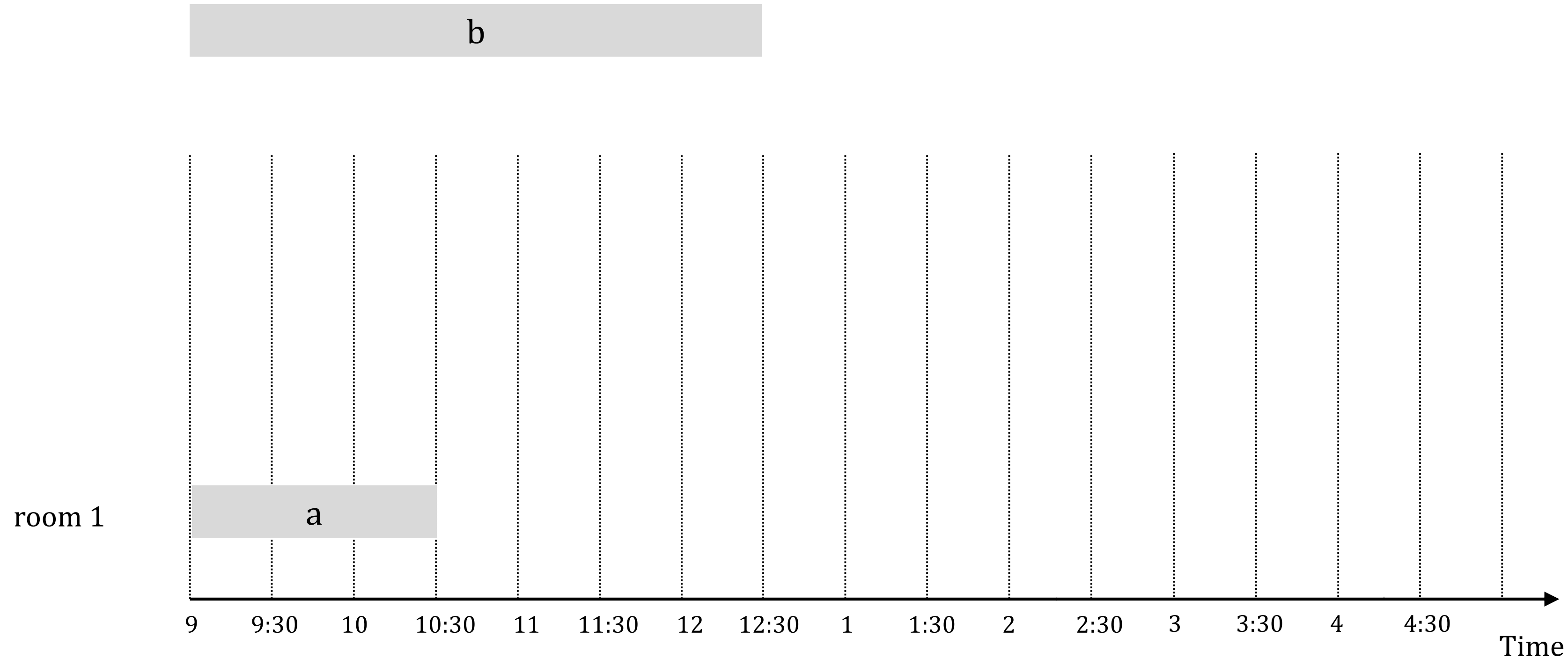




Interval Partitioning Problem Example

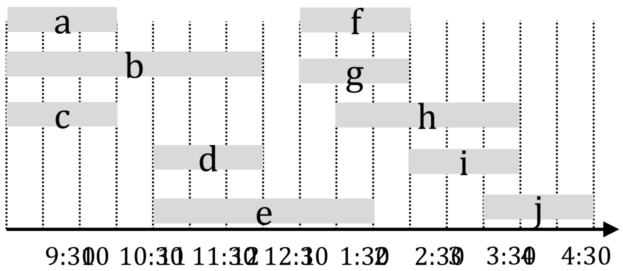
Example





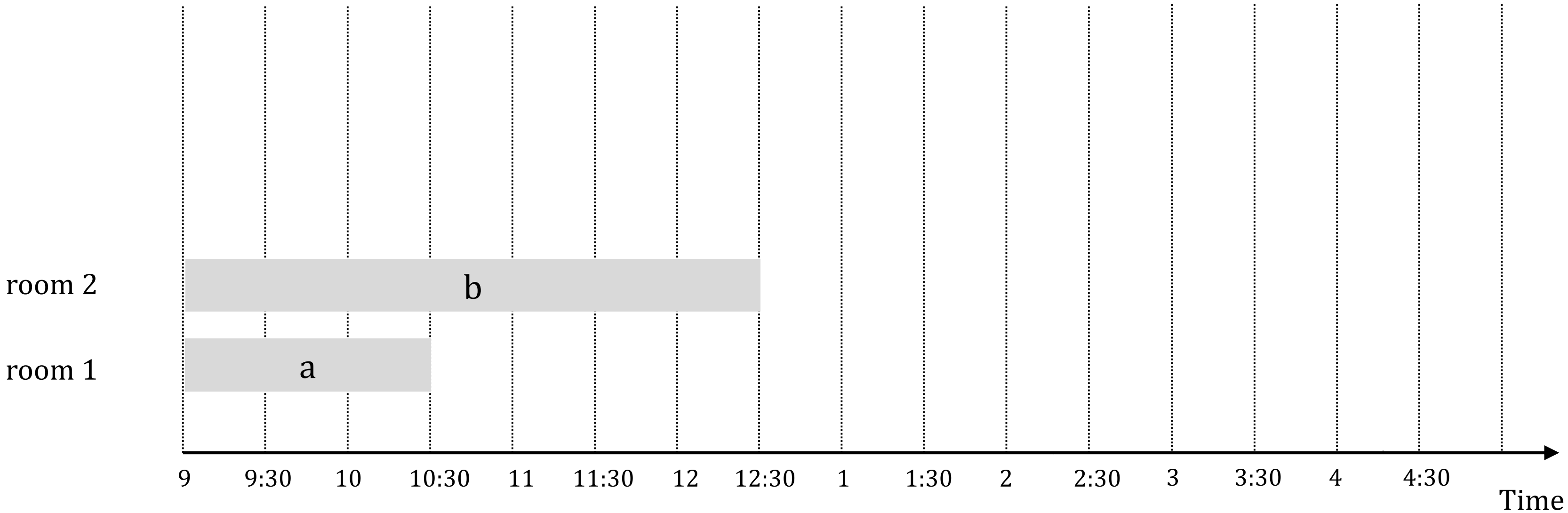
Interval Partitioning Problem Example

Example



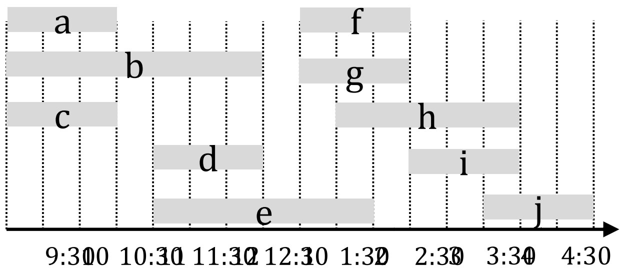
The algorithm only introduces room 2 when there are 2 intervals that overlap: in this case a, b.

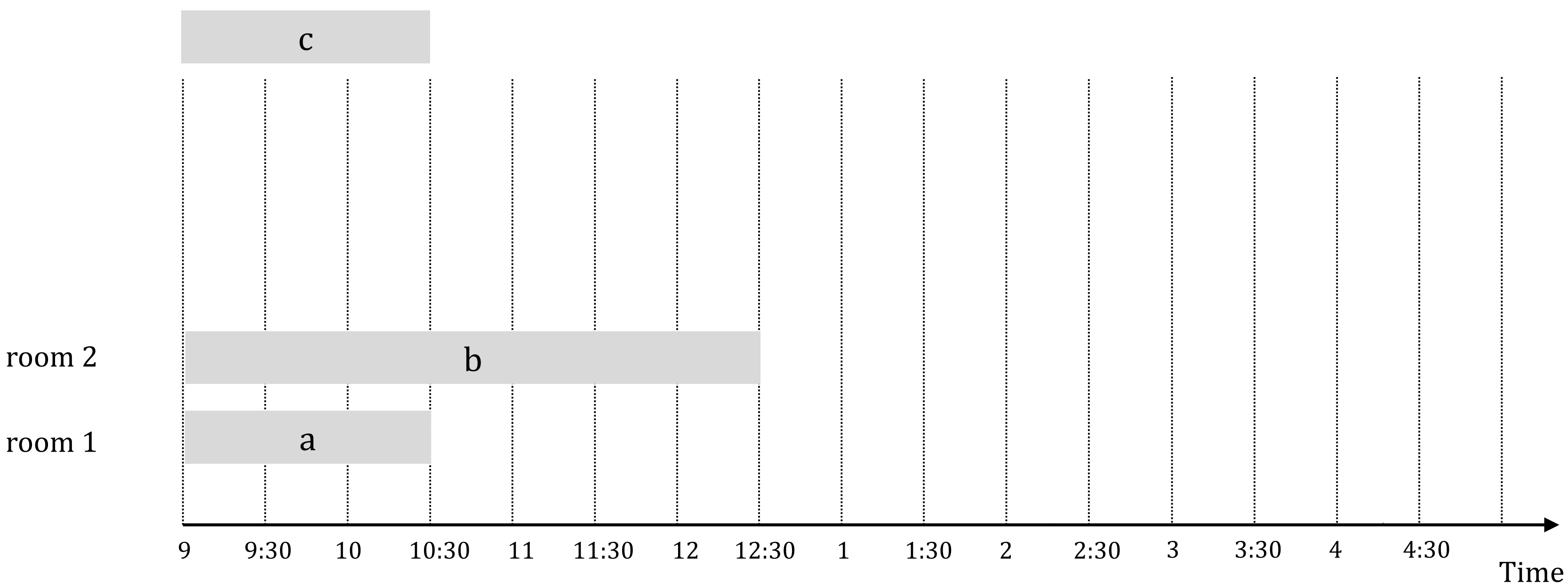
Depth is always the minimum it can be.



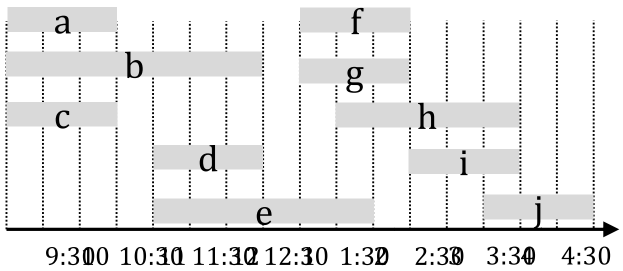
Interval Partitioning Problem Example

Example





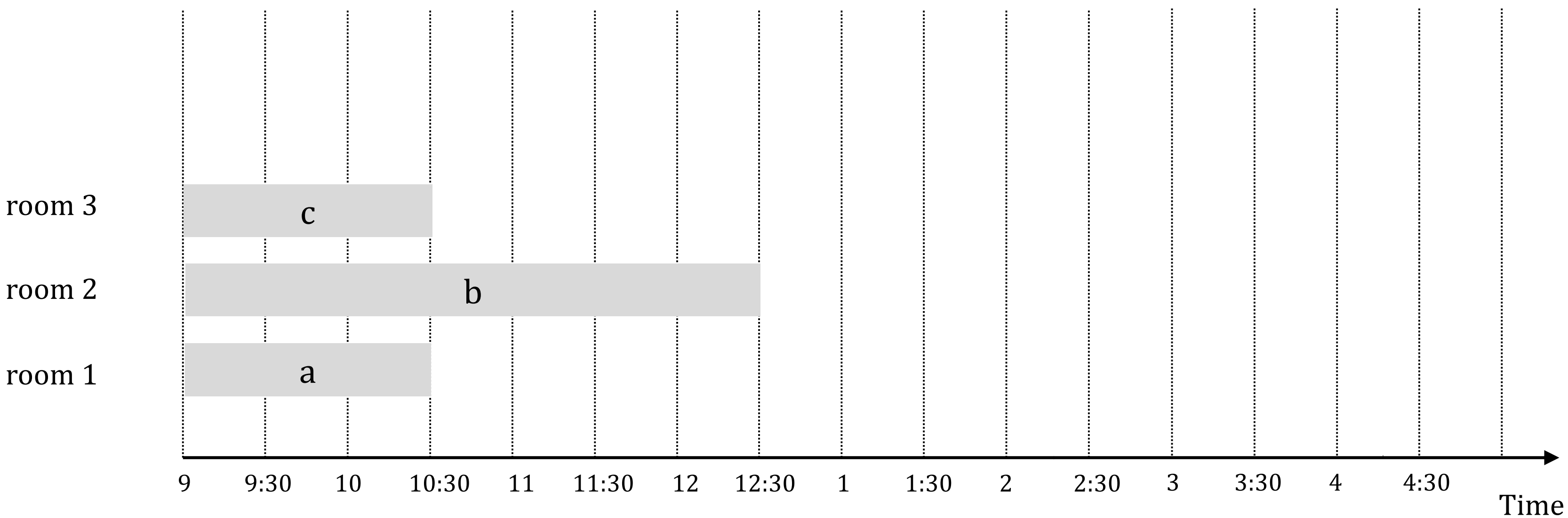
Interval Partitioning Problem Example



Example

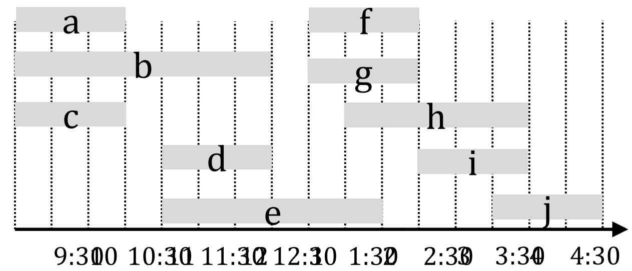
The algorithm only introduces room 3 when there are 3 intervals that overlap: in this case a,b,c.

Depth is always the minimum it can be.



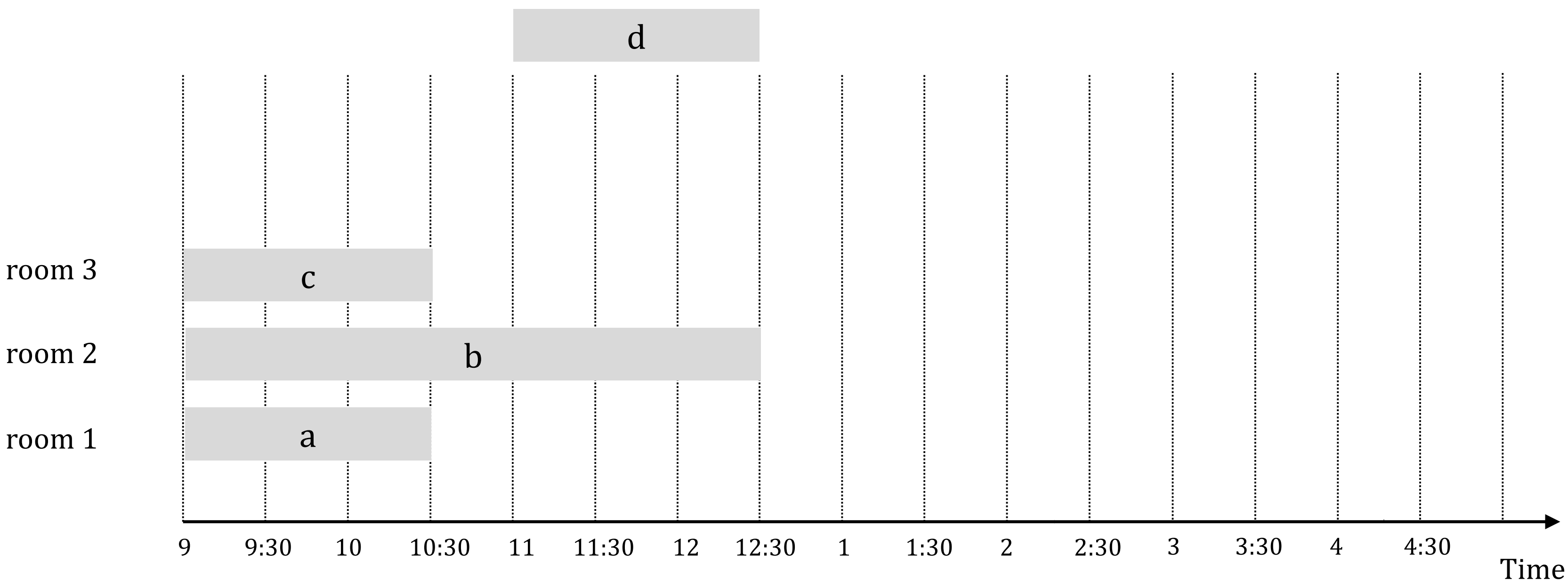
Interval Partitioning Problem Example

Example



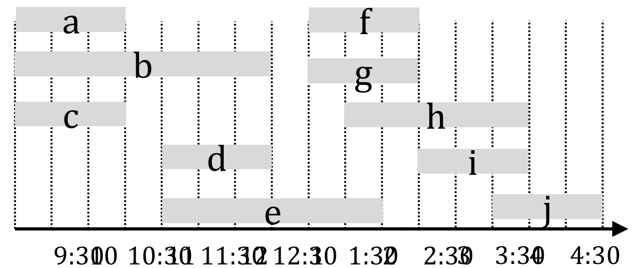
The algorithm only introduces room 3 when there are 3 intervals that overlap: in this case a,b,c.

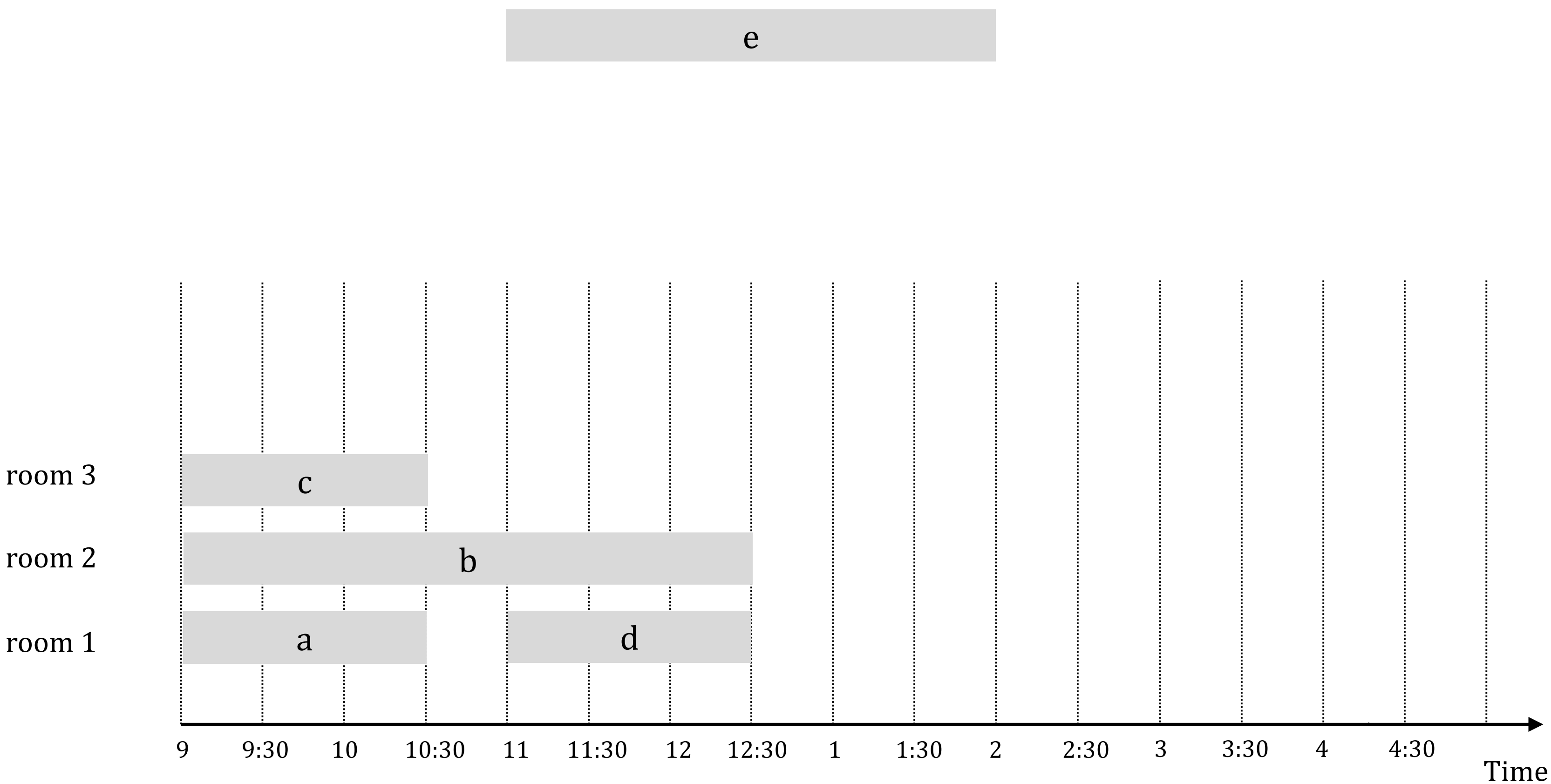
Depth is always the minimum it can be.



Interval Partitioning Problem Example

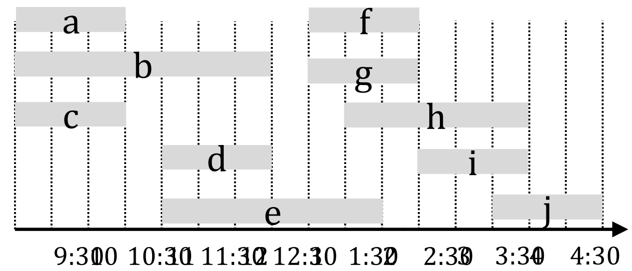
Example

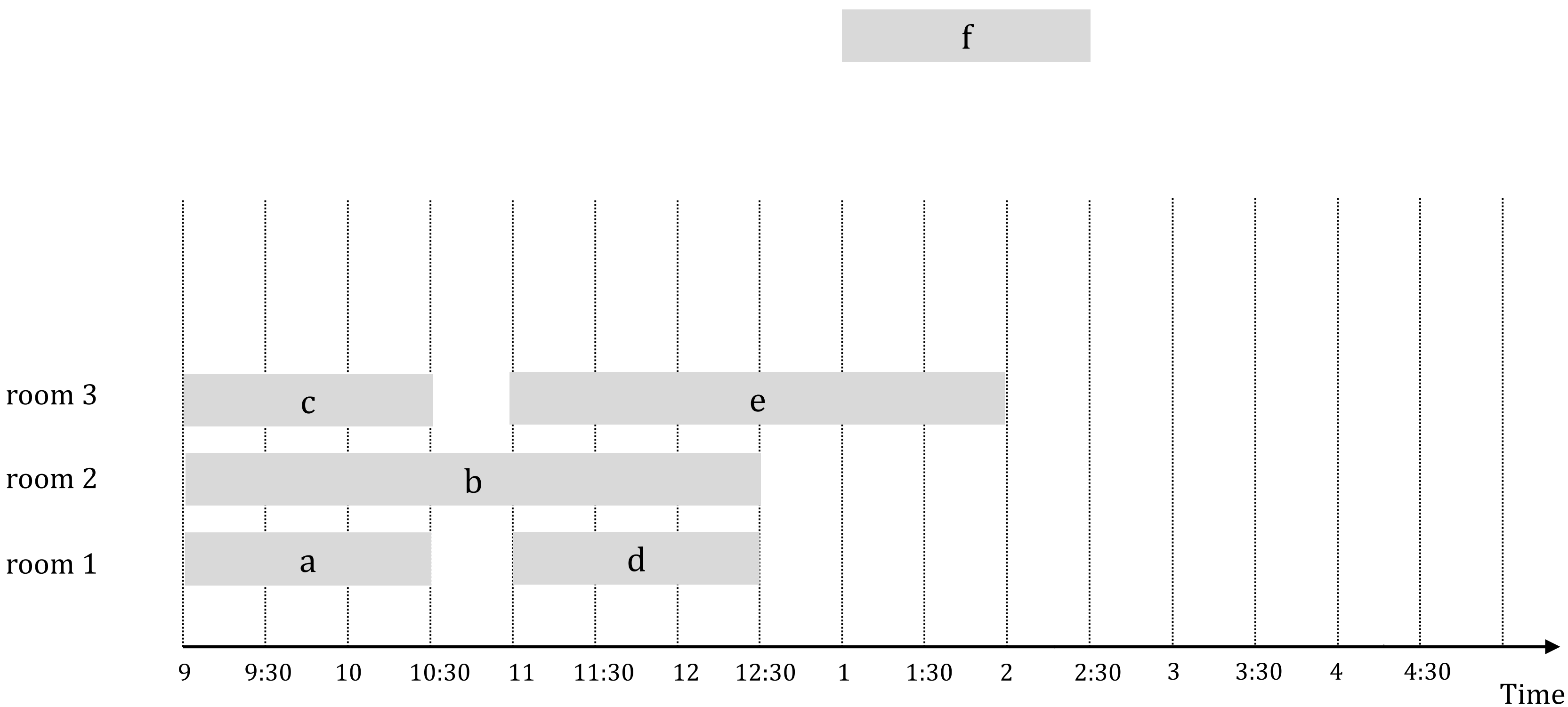




Interval Partitioning Problem Example

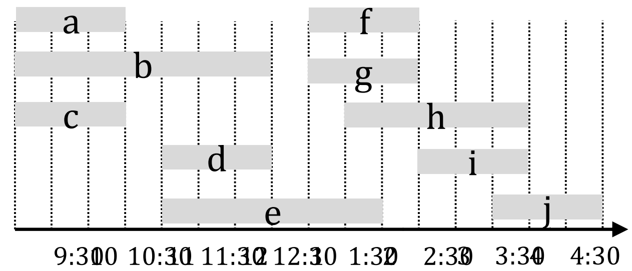
Example

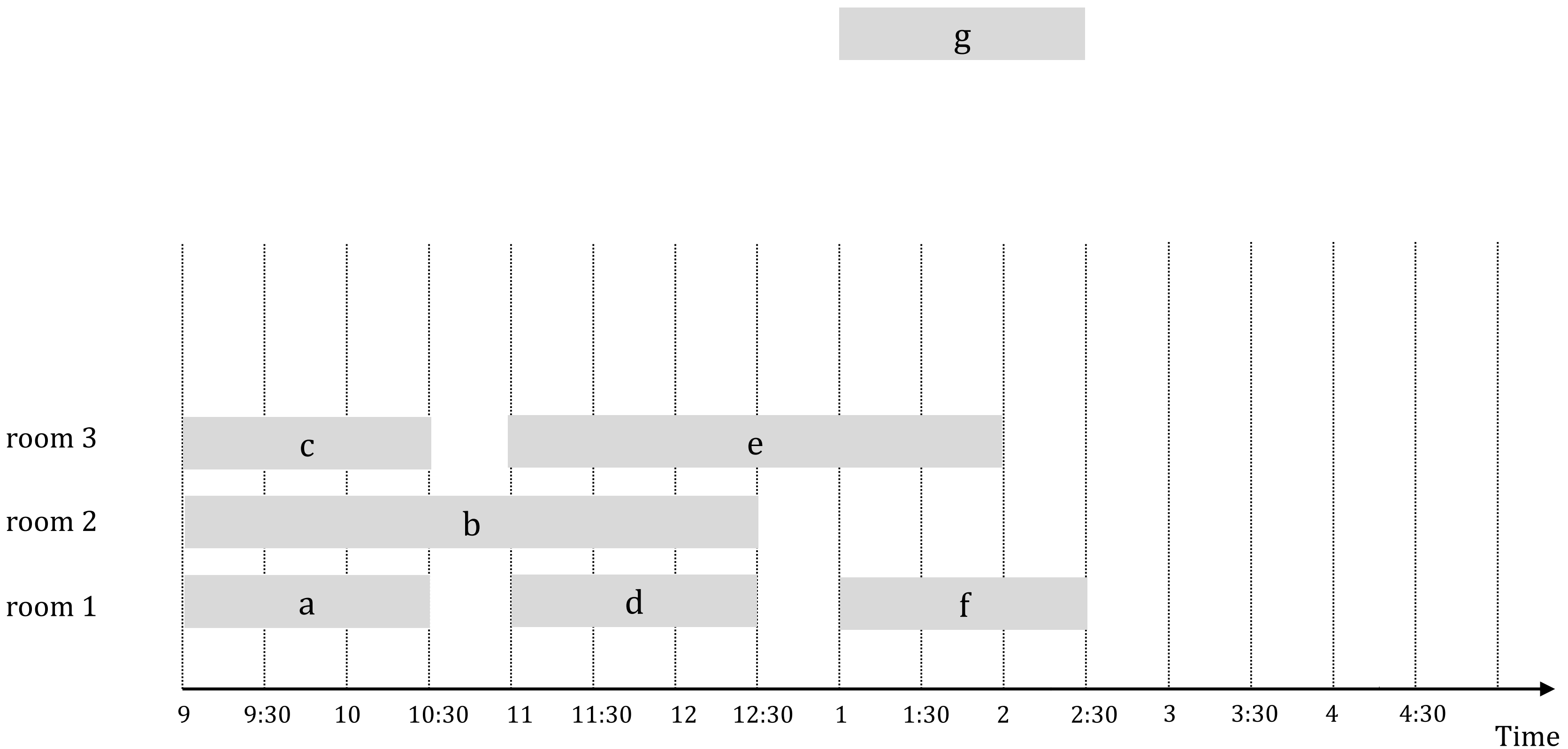




Interval Partitioning Problem Example

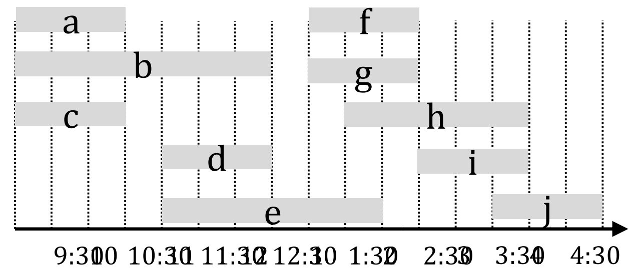
Example

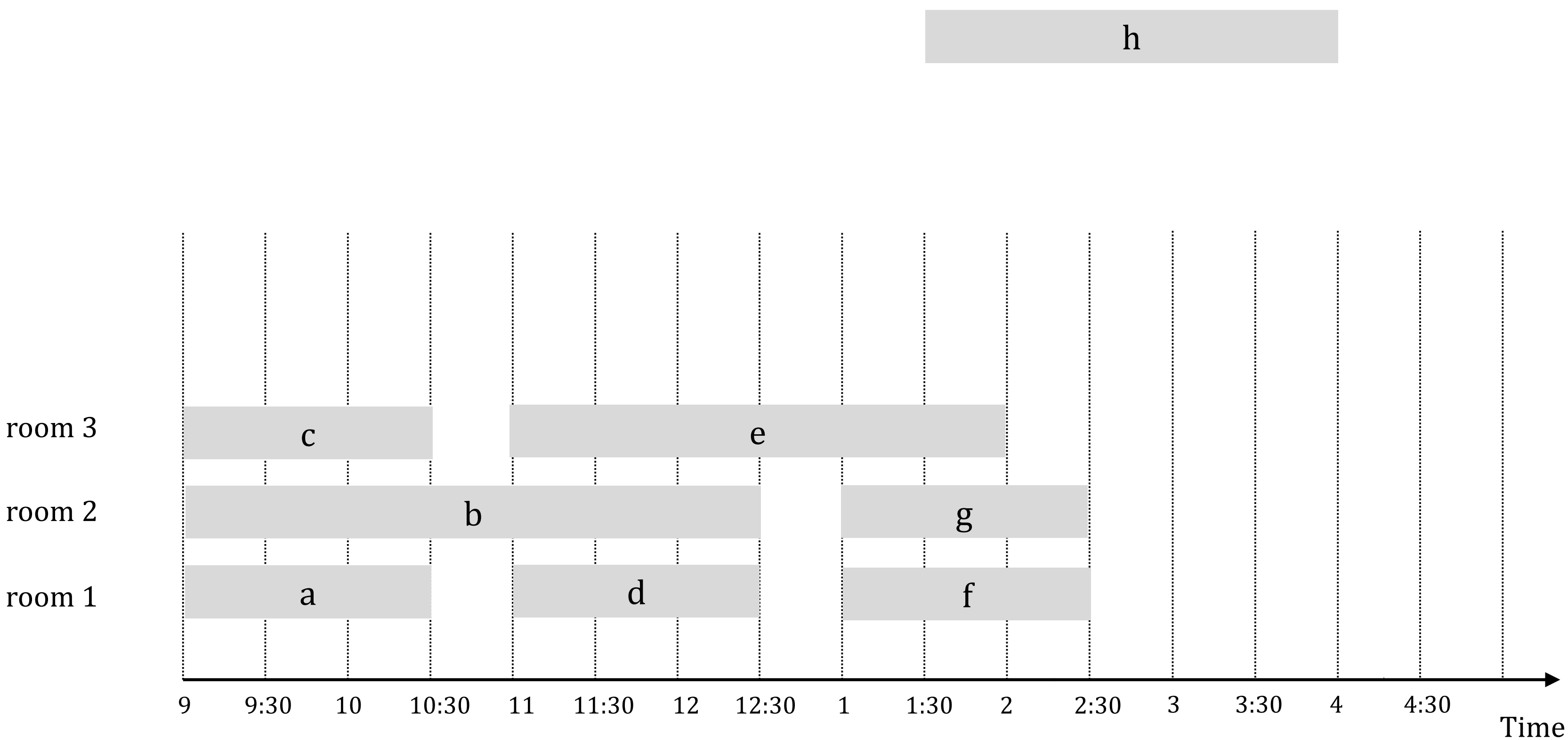




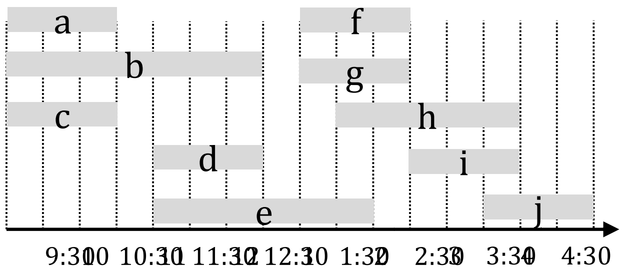
Interval Partitioning Problem Example

Example





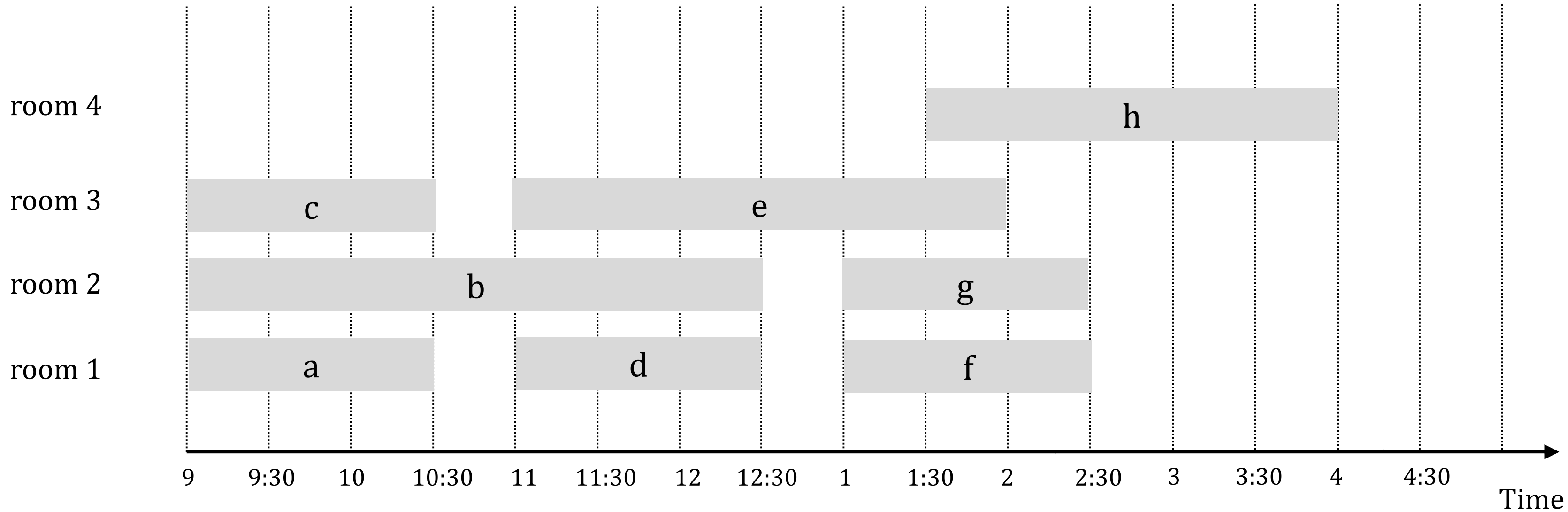
Interval Partitioning Problem Example



Example

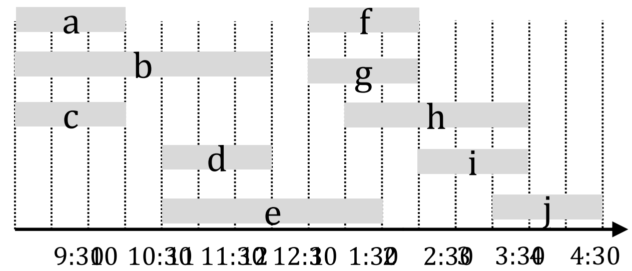
The algorithm only introduces room 4 when there are 4 intervals that overlap: in this case e, f, g, h.

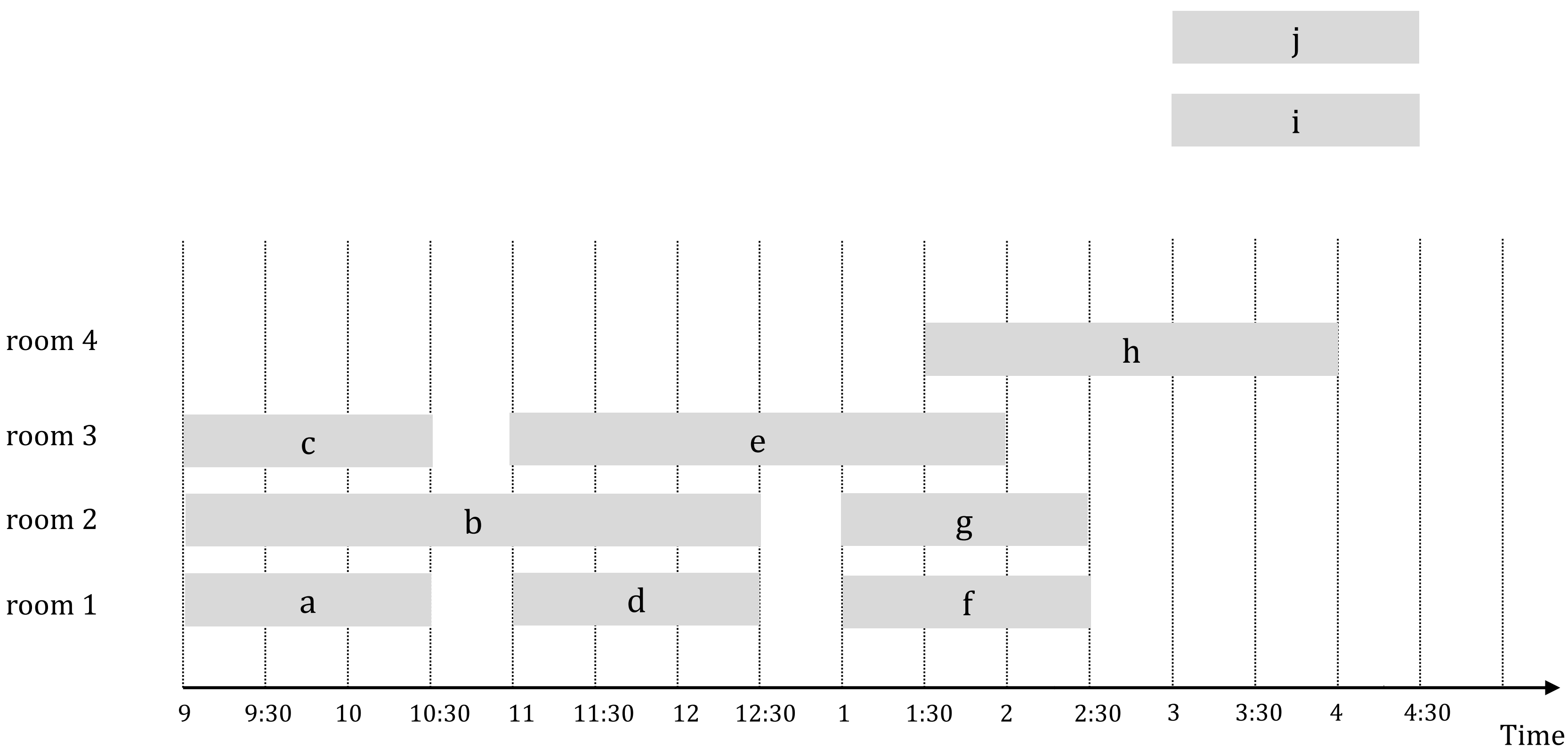
Depth is always the minimum it can be.



Interval Partitioning Problem Example

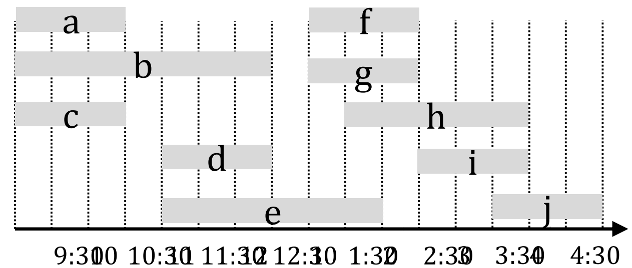
Example





Interval Partitioning Problem Example

Example:



Overall depth is 4, this is the minimum number of rooms needed to schedule the intervals, and the greedy algorithm has found such a schedule.