TDT4121 Introduction to algorithms

Assignment 5

Claudi Lleyda Moltó

October 2024

Greedy algorithms

- Intuitively, these are algorithms making the locally optimal choice at every stage.
- Greedy algorithms can be very varied, but there are some common patterns that help us work with them.

Greedy algorithms

- Intuitively, these are algorithms making the locally optimal choice at every stage.
- Greedy algorithms can be very varied, but there are some common patterns that help us work with them.

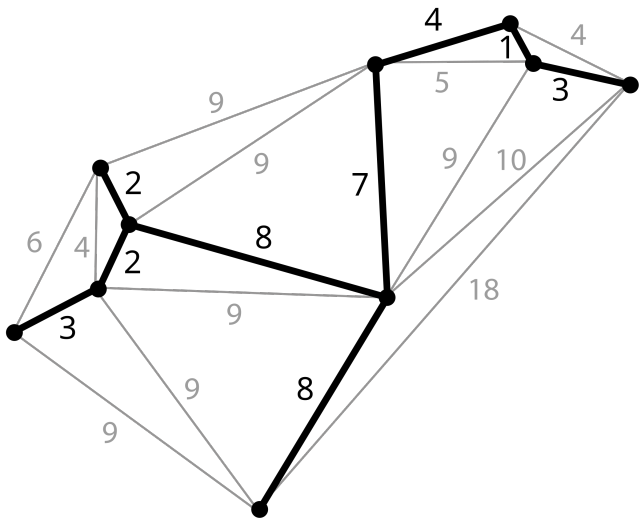
- Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as the solution of any other algorithm.
- Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
- Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that our algorithm always achieves this bound.

Tips seen in class slides, taken from Kevin Wayne.

Minimum spanning tree

Suppose we have a weighted graph, and we want to reduce its edges to the least possible, in the sense that

- We should have no redundant edges; the graph should be a tree
- The total cost of the edges must be minimized.
- The graph should remain connected.



Minimum spanning tree

- There exist different optimal algorithms to solve this problem.

Kruskal's algorithm

- Iterate over the sorted edges in order of increasing weight.
- If an edge can be added to the result graph without creating a cycle, add it. Otherwise skip it.

The algorithm stops naturally when result graph is fully connected.

Minimum spanning tree

- There exist different optimal algorithms to solve this problem.

Kruskal's algorithm

- Iterate over the sorted edges in order of increasing weight.
- If an edge can be added to the result graph without creating a cycle, add it. Otherwise skip it.

The algorithm stops naturally when result graph is fully connected.

Reverse-delete algorithm

- Iterate over the sorted edges in order of decreasing weight.
- If an edge can be removed without making the graph disconnected, remove it. Otherwise skip it.

The algorithm stops naturally when result graph is a tree.

- These symmetric algorithms find an optimal answer.

Minimum spanning tree

- We can use the following results to prove the previous algorithms work.

Cut property

Take a weighted graph G where all the edge costs are distinct, and take a non-empty proper subgraph S of G. Since G is connected, there exists at least an edge between S and $G \setminus S$. Let e be the least weighted of such edges.
Then any minimum spanning tree must contain e.

Minimum spanning tree

- We can use the following results to prove the previous algorithms work.

Cut property

Take a weighted graph G where all the edge costs are distinct, and take a non-empty proper subgraph S of G. Since G is connected, there exists at least an edge between S and $G \setminus S$. Let e be the least weighted of such edges.
Then any minimum spanning tree must contain e.

Cycle property

Take a weighted graph G where all the edge costs are distinct, and let C be any cycle in G, where e is the most expensive edge in said cycle.
Then e cannot belong to any minimum spanning tree of G.

Interval partitioning problem

- Suppose we are in charge of organising classrooms for lectures.
- You are given a list on n lectures with their starting and finishing times s_i and f_i,
- You would like to minimise the number of classrooms needed to hold the lectures, while ensuring that they never overlap.

Interval partitioning problem

- Suppose we are in charge of organising classrooms for lectures.
- You are given a list on n lectures with their starting and finishing times s_i and f_i,
- You would like to minimise the number of classrooms needed to hold the lectures, while ensuring that they never overlap.

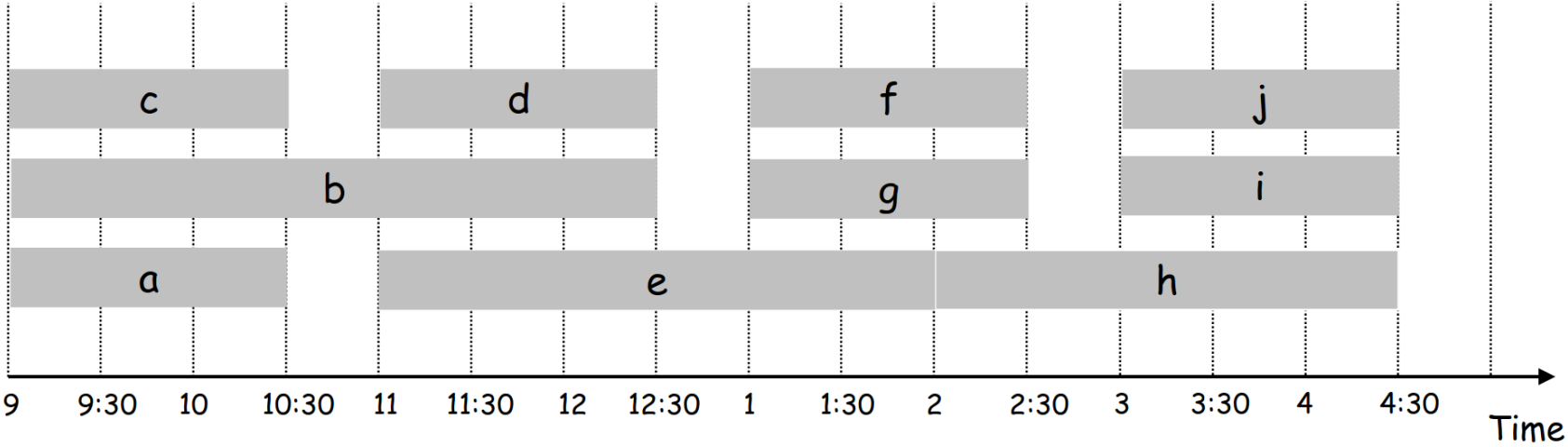


Figure seen in class slides, taken from Kevin Wayne.

Interval partitioning problem

```
function ScheduleClasses(List lectures; Integer n)
    Sort lectures by starting time
    d ← 0
    for j in 1, ..., n do
        for i in 1, ..., d do
            if lecture j fits in classroom i then
                schedule lecture j in classroom i
        if lecture j has no been scheduled then
            d ← d + 1
            schedule lecture j in classroom d
```

- Note how the algorithm avoids overlapping lectures in the same classroom.
- In the recitation lectures we saw how the number of classrooms allocated is at most to the depth of the problem.

Interval partitioning problem

- Note how if there are d lectures that request the same time instant, then we know that we need at least d classrooms to hold them without overlapping.
- The maximal of these values is called the *depth* of the problem. It is a lower bound to how many classrooms we need.

Interval partitioning problem

- Note how if there are d lectures that request the same time instant, then we know that we need at least d classrooms to hold them without overlapping.
- The maximal of these values is called the *depth* of the problem. It is a lower bound to how many classrooms we need.

- In fact, we proved with the previous algorithm that the optimal answer for a problem of depth d requires exactly d classrooms.
- This hints at an intrinsically local structure in the problem, where assigning a classroom to a lecture does not have broad consequences on the overall schedule.
- When designing another algorithm to solve this problem, we can use this result to check that our solution achieves this bound.