Algorithmic Strategies 2024/25

Week 6 – Greedy Algorithms

· U C ·

UNIVERSIDADE DE COIMBRA

Outline

1. Introduction
2. Fractional knapsack problem
3. Activity-selection problem

Reading about problem solving with greedy algorithms

- J. Erickson, Algorithms, Chapter 4
- T. Cormen et al., Introduction to Algorithms, Chapter 16
- J. Edmonds, How to think about algorithms, Chapter 16
- J. Kleinberg and E. Tardos, Algorithm design, Chapter 4
- S. Dasgupta et al., Algorithms, Chapter 5

Greedy algorithms

- The greedy strategy: Make whatever choice looks best
- They are much simpler to implement than divide-and-conquer or dynamic programming algorithms
- However, they do not always yield optimal solutions for all problems, but they they may provide good approximations.

Introduction

When does a greedy algorithm solve a given problem?

- Optimal substructure property: An optimal solution to a problem contains within it optimal solutions to sub-problems.

- Greedy-choice property: A globally optimal solution can be constructed by making a locally optimal (greedy) choice.

Combination of optimal substructure and greedy choice:

1. The optimal solution contains the first best choice (greedy choice)
2. By removing the best choice from the optimal solution, we obtain the optimal solution for the subproblem without that choice (optimal substructure).
3. Use induction to prove for the remaining cases.

Greedy exchange argument:

- Assume that the first greedy choice is not in the optimal solution.
- Then there is another first choice in the optimal solution.
- Show that by exchanging that choice by the greedy choice, the solution does not worsen or contradits the step above.

Fractional knapsack problem

- A thief is robbing a jewellery and he finds n ingots, each of which is made of a different precious metal. Assume that the i-th ingot has value v_i and weight w_i.
- He cannot carry more than W kg in his knapsack, and he wants to take as much value as possible.
- He has a knife, so he can cut each ingot into small pieces. How much of each ingot should he take?

This is the fractional knapsack problem!

How would you solve the problem, intuitively?

Fractional knapsack problem

Example

- Ingot 1: 60 €, 10 kg
- Ingot 2: 100 €, 20 kg
- Ingot 3: 120 €, 30 kg
- W = 50 kg

Ingot 1, Ingot 2 and 2/3 of Ingot 3 gives 240 € and 50 kg.

Fractional knapsack problem

Function *knapsack*(W)

```
sort ingots into nonincreasing order by v_j/w_j
w = 0, v = 0, j = 1
while w < W do
    if w + w_j ≤ W then                          {Take ingot j}
        w = w + w_j
        v = v + v_j
    else                                         {Take a fraction of ingot j}
        v = v + v_j · (W − w)/w_j
        w = W
    j = j + 1
return v
```

Is the total value v optimal?

Fractional knapsack problem

We need to show that the algorithm is correct in two steps:

1. Optimal substructure property
2. Greedy choice property

Optimal substructure

- Let S be the optimal load for constraint W.
- By removing a fraction ϵ from ingot j in S, we have the optimal load for the remaining ingots and $(1 - \epsilon)$ fraction of ingot j for constraint $W - \epsilon w_j$.

Why?

Fractional knapsack problem

1. Let S be the optimal load for the subproblem of the first j ingots, with total value v and total weight $w \leq W$.

2. Then, S without a fraction ϵ of ingot j, with total value $v - \epsilon v_j$ and total weight $w - \epsilon w_j$, is optimal for the subproblem with the remaining $j - 1$ ingots and $(1 - \epsilon)$ fraction of ingot j for constraint $W - \epsilon w_j$.

Sketch of the proof (by contradiction)

- (negate 2.) Assume that there exists another load for the subproblem above with total value $v' > v - \epsilon v_j$ and total weight $w' \leq W - \epsilon w_j$.

- (contradict 1.) Then, it also exists a load with total value $v' + \epsilon v_j > v$ and weight $w' + \epsilon w_j \leq W$!

Fractional knapsack problem

Greedy choice property

- The item with the highest ratio v_j/w_j is in the optimal load.

Why?

Fractional knapsack problem

(By contradiction) Let v_j/w_j be the highest ratio and is not in the optimal load.

1. Assume that the knapsack is not full. Then take some of ingot j and get a higher value than the optimal! $\Rightarrow$ contradition!

2. Assume that the knapsack is full. Then, there must exist some ingot $k \neq j$ in the knapsack with

$$\frac{v_k}{w_k} < \frac{v_j}{w_j}$$

- Exchange α kg of ingot k by α kg of ingot j. This would increase the optimal value by

$$v_j \cdot \frac{\alpha}{w_j} - v_k \cdot \frac{\alpha}{w_k} > 0 \quad \Rightarrow \quad \text{contradition}$$

Fractional knapsack problem

Implication of the two properties:

1. (greedy choice) Take the ingot j with the highest ratio v_j/w_j. If $w_j > W$, take part of ingot j that fills the knapsack.

2. (optimal substructure) Now, we have a problem of $W - w_j$ and goto 1 or stop if the knapsack is full.

Fractional knapsack problem

What if the thief does not have a knife?

- Ingot 1: 60 €, 10 kg
- Ingot 2: 100 €, 20 kg
- Ingot 3: 120 €, 30 kg
- W = 50

Pick first the ingots that have larger ratio value/weight:

Ingot 1 and Ingot 2 gives 160 €

But the optimal is to take Ingot 2 and Ingot 3 which gives 220 €!?

Fractional knapsack problem

- If the thief cannot cut the ingots, then it corresponds to the knapsack problem (take or not take the complete ingot).

- Although it has optimal substructure, the greedy choice property does not hold. Still, the greedy algorithm gives an approximation to the optimal value.

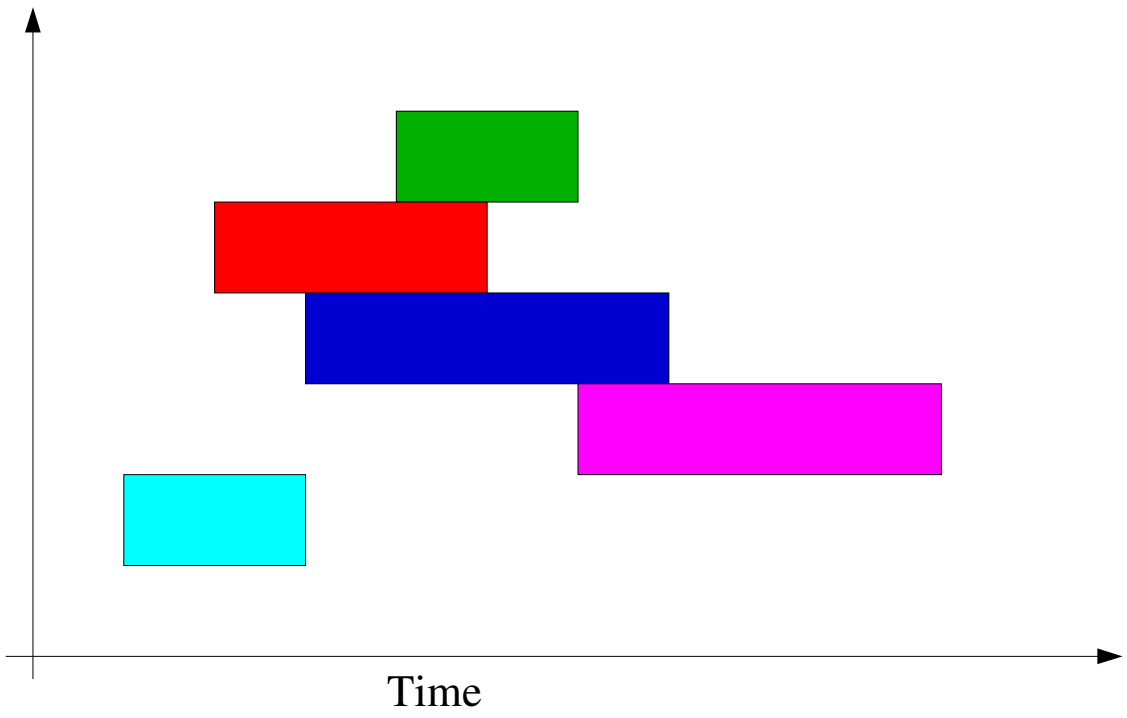
- In fact, it becomes an NP-hard problem and no greedy algorithm is known that can provide an otimal solution.

Activity-selection problem

- A set $S = \{1, \ldots, n\}$ of activities needs a room. The room can only be used by one activity at a time.
- Each activity i takes place during the interval $[s_i, f_i)$.
- Two activities are compatible if their intervals do not overlap.

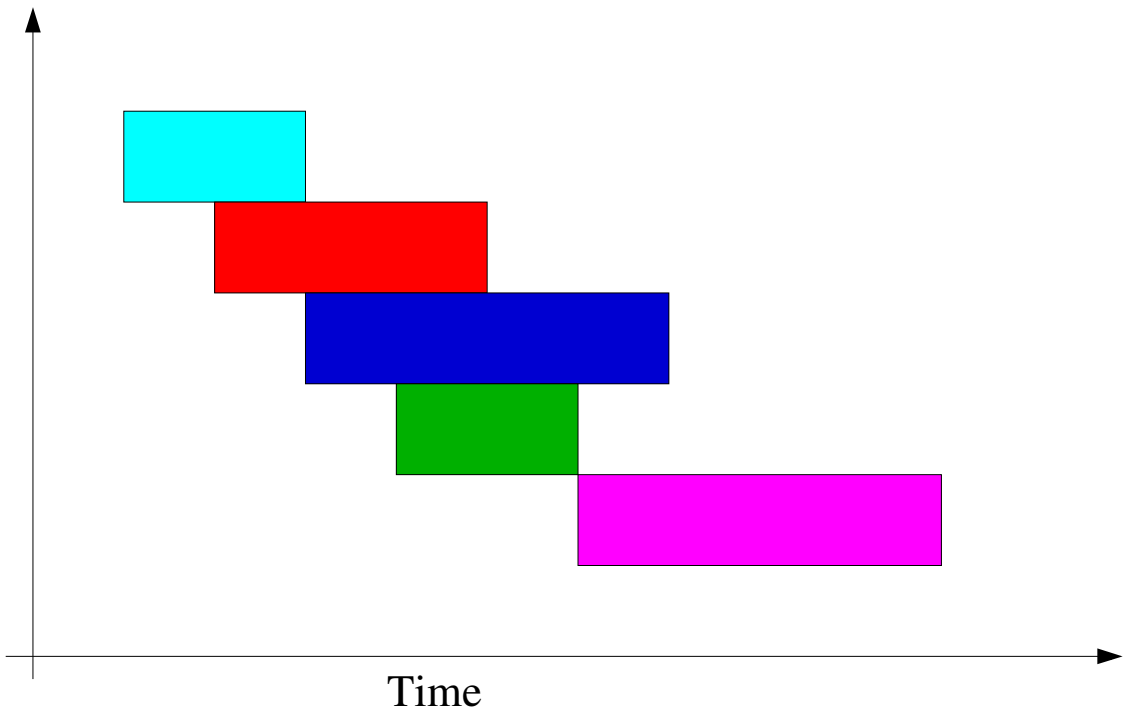
Select the maximum-size set of mutually compatible activities

Activity-selection problem



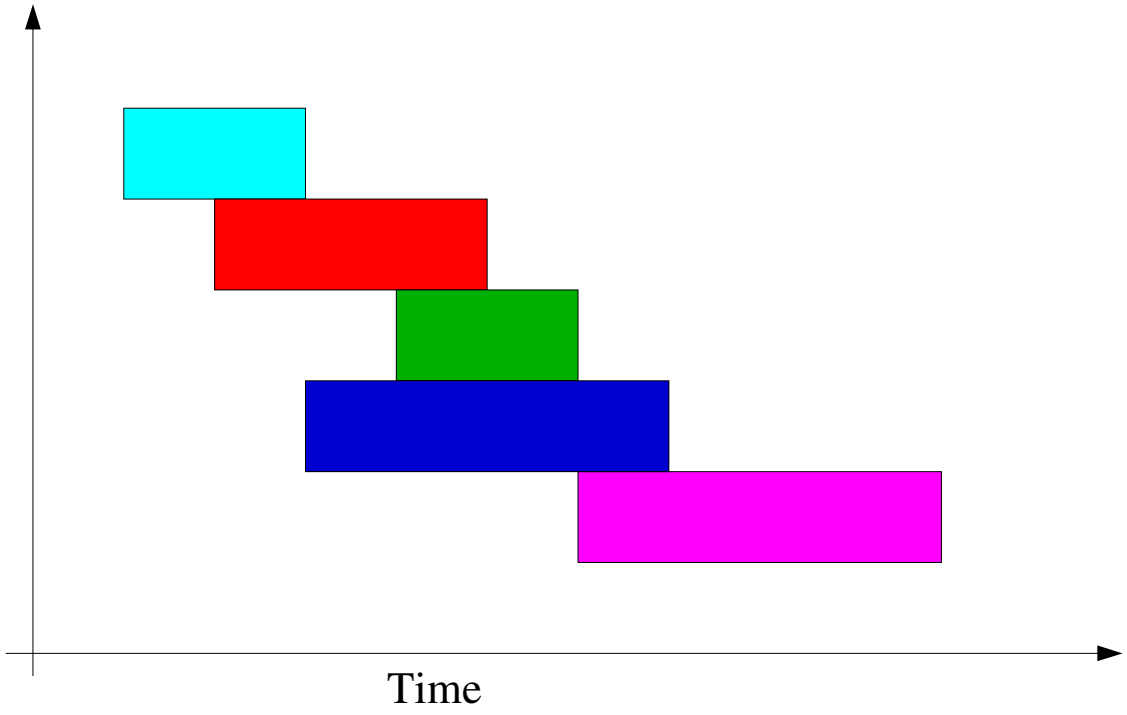
How to solve the problem?

Activity-selection problem



Sorting in non-decreasing order by starting times?

Activity-selection problem



Sorting in non-decreasing order by finishing times?

Activity-selection problem

Function *selection*(S)

sort S into nondecreasing order by finishing times f_i
$A = \{1\}$
$j = 1$
for i in S **do**
  if $s_i \geq f_j$ **then**
    $A = A \cup \{i\}$
    $j = i$
return A

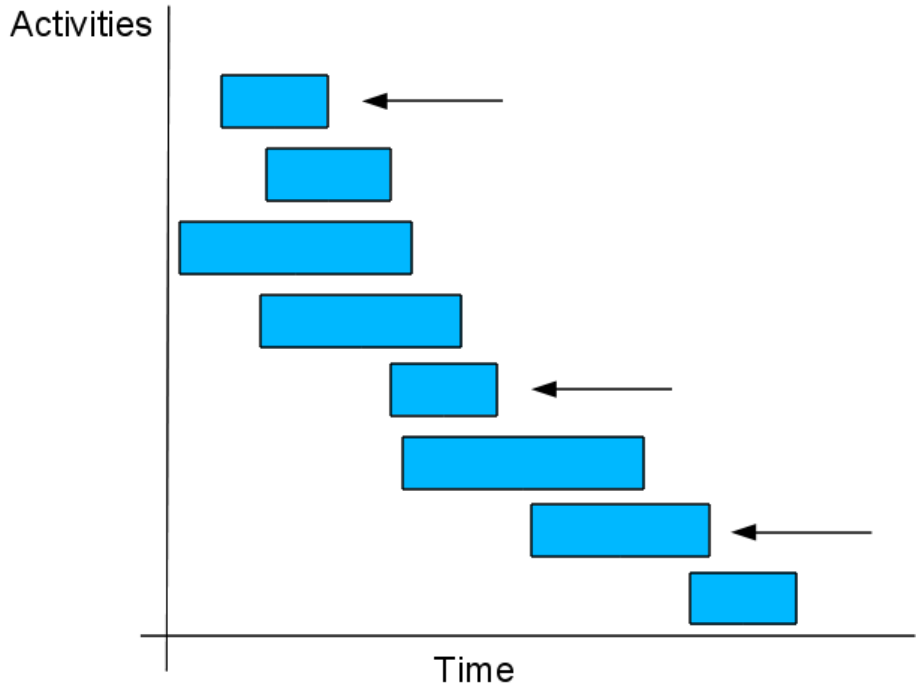
- Greedy choice: choose the next earliest finishing time activity that does not overlap
- Why it works?

Activity-selection problem

Optimal substructure

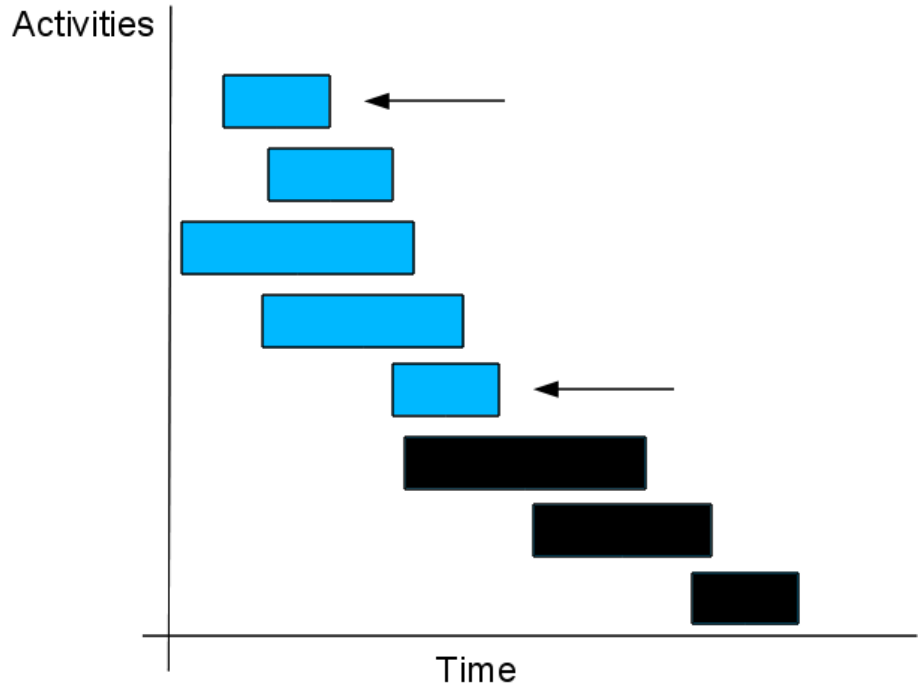
- If A is an optimal solution, then, by removing activity j, the optimal solution A' is obtained for the remaining activities that do not overlap with it.

Activity-selection problem



Optimal solution for the problem

Activity-selection problem



Optimal solution for the sub-problem

Activity-selection problem

Optimal substructure

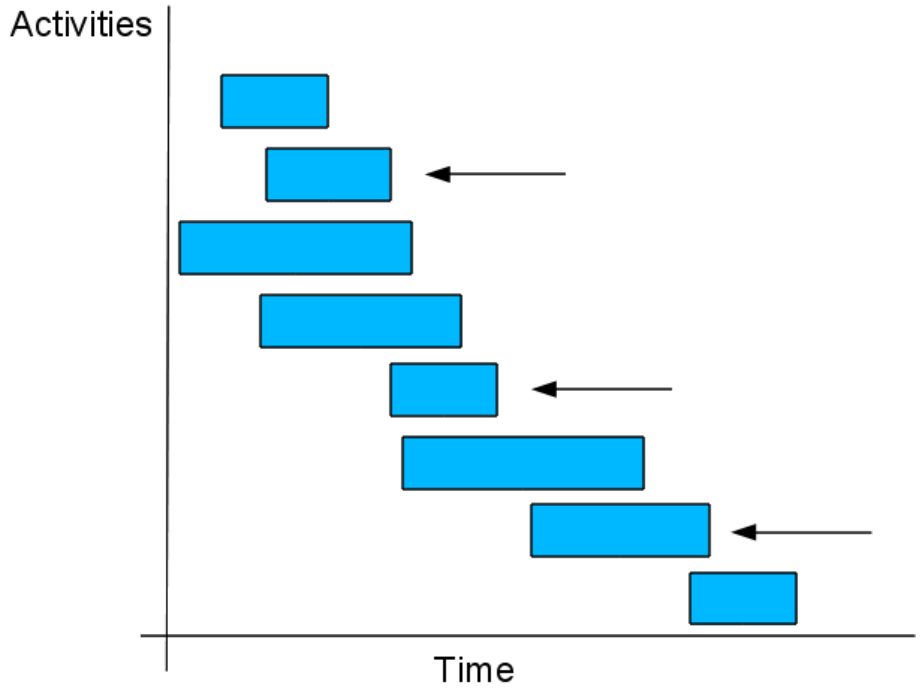
- If A is an optimal solution, then by removing activity j the optimal solution A' is obtained for the remaining activities that do not overlap with it.

- If A' is not optimal, then there exists an optimal solution B that contains larger number of activities that do not overlap with j. But then, this would imply that $|B \cup \{j\}| > |A| \Rightarrow$ contradiction!

Greedy choice property

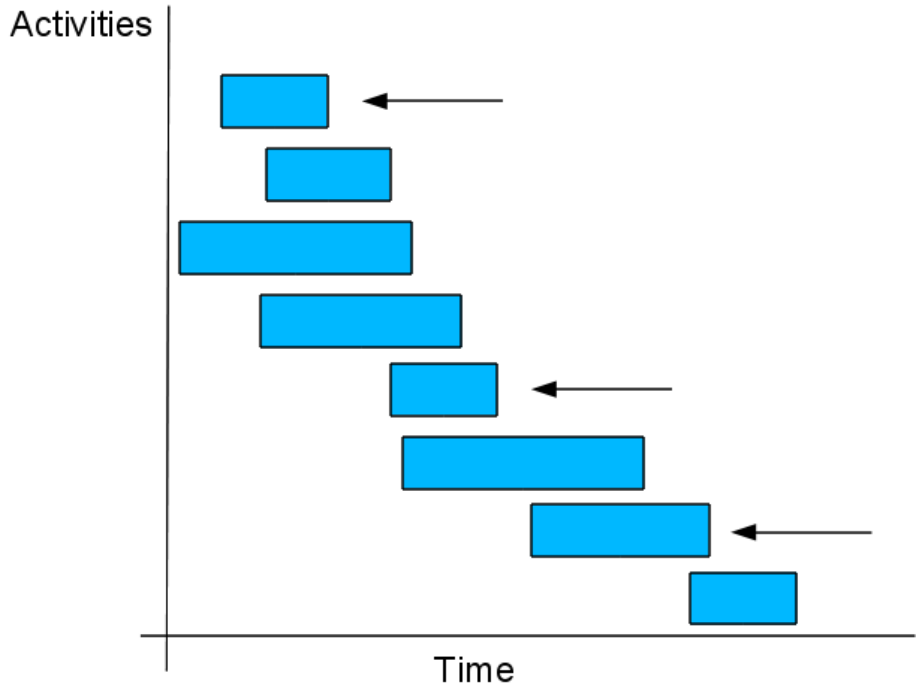
- Assume that A is optimal and its first activity is $k > 1$.
- Then, there is another optimal subset B of activities that begins with greedy choice (activity 1) and has the same size:
  - Obtain B by exchanging k by 1
  - Since $f_1 \leq f_k$, the activities in B do not overlap
  - Then, A and B must have the same size!

Activity-selection problem



Optimal solution by selecting 2 as the first choice

Activity-selection problem



Optimal solution by selecting 1 as the first choice

Activity-selection problem

Implication of the two properties:

1. (greedy choice) Choose the activity with earliest finishing time and ignore all activities that are overlapping.

2. (optimal substructure) Now, we have a smaller problem with the remaining activities. Goto 1 or stop if there are no more activities.