

Problem Solving Greedy Algorithm

Marie Pelleau
marie.pelleau@univ-cotedazur.fr

Greedy Algorithm

Definition

- A **greedy algorithm** is an algorithm that follows the principle of making a locally optimal choice at each step, with the hope of reaching a globally optimal result
- In cases where the algorithm does not always provide the optimal solution, it is called a **greedy heuristic**
- The method used to make the choice is sometimes referred to as a **greedy strategy**

Greedy Algorithm

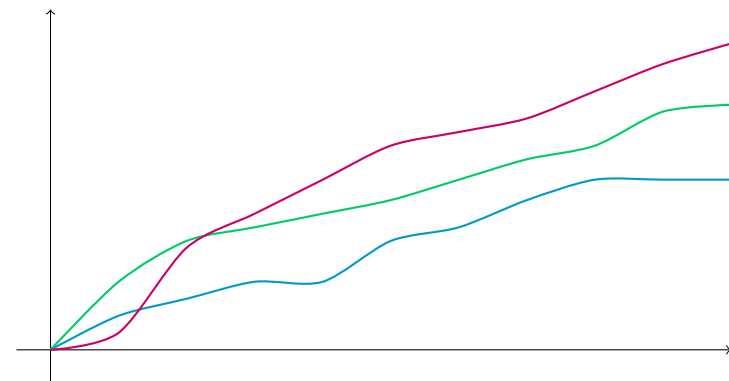
Change with the Fewest Coins

Greedy strategy: At each step, return the coin with the largest value that is smaller than the remaining amount to be returned

- To return 37 cents
 - The largest coin is 20, so return 20, and 17 cents remain
 - The largest coin is 10, so return 10, and 7 cents remain
 - The largest coin is 5, so return 5, and 2 cents remain
 - Return 2
 - Thus, you have returned $20+10+5+2$
-
- In the European coin system, the greedy algorithm always gives an optimal solution
 - However, in the coin system (1, 3, 4), the greedy algorithm is not optimal (for 6: it gives $4+1+1$, while $3+3$ is optimal)

Greedy Algorithm

- Principle: be efficient
- Difficulty: finding the good strategy, or an efficient one



Knapsack Problem

Description

You have:

- A backpack with a weight limit
- A set of objects, each object o_i has:
 - A weight: p_i
 - A value: v_i

Which objects should be taken to maximize the total value carried while respecting the weight constraint?

- The total value of the selected objects is maximized
- The total weight of the selected objects is less than or equal to the backpack's weight limit

Knapsack Problem

You have bags of precious metals o_i and you can take as much as you want, but no more than p_i

For each bag, calculate the value per gram

- 1 Take the bag with the highest value per gram and fill the backpack with as much of that bag as possible
- 2 If the weight limit of the backpack is reached, stop
- 3 Otherwise, take the entire content of the bag, eliminate that bag, and repeat step 1

This strategy is optimal

Knapsack Problem

Optimality Proof

- Define the efficiency of an object o_i : as the value per gram
 - Suppose there is an optimal solution that does not take one gram of an object o_i but does take one gram of an object o_k with lower efficiency than o_i
 - By replacing 1g of o_k with 1g of o_i the solution improves
- ⇒ Contradiction

Knapsack Problem

- Putting bags of precious metal powder is equivalent to accepting cutting objects
- If we can't cut objects, what do we do?
- The problem becomes difficult
- We proceed "as if"
- We calculate the efficiency and select the objects based on their efficiency while respecting the weight constraint

Knapsack Problem

Example

We have a backpack with a maximum capacity of 15kg and the following items:

- o_1 of value 10 and weight 9kg
- o_2 of value 7 and weight 12kg
- o_3 of value 1 and weight 2kg
- o_4 of value 3 and weight 7kg
- o_5 of value 2 and weight 5kg

Knapsack Problem

Modelization

- How do we represent this problem?
- What are the variables (the unknowns)?
- How do we express the constraints?

This is the modelization: the mathematical representation of the problem

Knapsack Problem

Variables

- We associate each item with a 0-1 variable (it only takes values 0 or 1)
- It is a membership variable for the backpack
- If the item is taken, the variable is 1, otherwise it is 0

Model

- The value and weight of an item are given data, so for item o_i , we have the value v_i and the weight p_i
- The membership variable for the backpack is x_i
- The maximum weight of the backpack is W

Knapsack Problem

Constraints

- $\max \sum_{i=1}^n v_i x_i$ the objective
- $\sum_{i=1}^n p_i x_i \leq W$ sum of weights less than or equal to the maximum weight

Heuristic

When the greedy algorithm does not always give an optimal solution, it is called a **heuristic method**

- It comes from the ancient Greek *eurisko*, meaning "I find"
 - A heuristic is an algorithm that quickly (in polynomial time) provides a feasible solution, not necessarily optimal, for an NP-hard optimization problem
 - A heuristic is an approximate method that does not always provide the exact solution
-
- We often refer to a heuristic method
 - Generally, a heuristic is designed for a specific problem, relying on its particular structure

Activity Selection Problem

Description

- We have a gym where multiple events take place: we want to fit in as many events as possible, knowing that two events cannot happen at the same time (there is only one gym)
- An event i is characterized by a start time s_i and an end time e_i
- Two events are compatible if their time intervals do not overlap

Question: How can we schedule the maximum number of events?

Evaluating a Heuristic

Practical or Empirical Criterion

The approximate algorithm is implemented, and the quality of its solutions is evaluated in comparison to optimal solutions (or the best-known solutions) on a *benchmark* (instances of the same problem accessible to all).

Mathematical Criterion

It must be shown that the heuristic guarantees performance, it is interesting to demonstrate a probabilistic guarantee when the heuristic often, but not always, provides good solutions.

Remarks

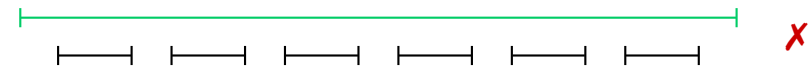
- Empirical and mathematical criteria can be contradictory
- Often, the guaranteed mathematical solution is not practical in real-life situations

Activity Selection Problem

Strategy 1

Sort the events by increasing start time

Counterexample

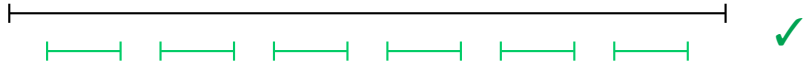


Activity Selection Problem

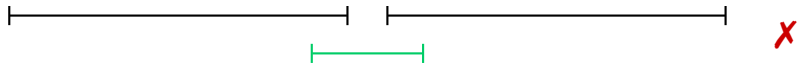
Strategy 2

Sort the events by increasing duration

Example



Counterexample

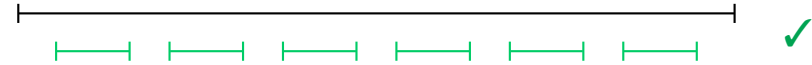


Activity Selection Problem

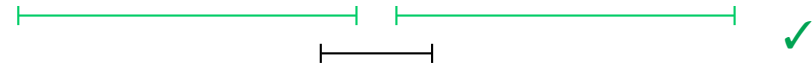
Strategy 3

Sort the events by increasing end time

Example



Example



Activity Selection Problem

Strategy 3

Sort the events by increasing end time

Property

This greedy algorithm is optimal

Activity Selection Problem

Proof by Induction

- Let f_1 be the earliest finishing event, let's show that there exists an optimal solution containing this event
- Let an arbitrary optimal solution be $O = f_{i_1}, f_{i_2}, \dots, f_{i_k}$ where k is the maximum number of events that can take place
- There are two possibilities:
 - either $f_{i_1} = f_1$
 - or $f_{i_1} \neq f_1$, we can replace f_{i_1} with f_1 because it finishes before any other event, and since f_{i_2} did not overlap with f_{i_1} , f_{i_2} will not overlap with f_1 either
Therefore, we can find an optimal solution with f_1 as the first event
- Then, we only consider the events that do not overlap with f_1 , and we repeat the process on the remaining events, hence the proof by induction

Exercise: Activity Selection

Description

- We have a gym where multiple events take place: we want to **maximize the gym's usage time**, knowing that two events cannot happen simultaneously (there is only one gym)
- An event i is characterized by a start time s_i and an end time e_i
- Two events are compatible if their time intervals do not overlap

Question: How can we maximize the gym's usage time?