# Greedy algorithms
# Advanced Algorithms

Master DSC/MLDM/CPS2

Amaury Habrard

amaury.habrard@univ-st-etienne.fr

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet Saint-Étienne
amaury.habrard@univ-st-etienne.fr

Semester 1

## 1 Avoiding the lack of petrol

## Avoiding the lack of petrol. 3

The algorithm is in $O(m \log m)$ if the stations are not sorted in distance increasing order, or linear in $m$ if the stations are already sorted.

```
$S \leftarrow \{d_0\}$;
$cur \leftarrow d_0$;
$i \leftarrow 1$;
while $i \leq m$ do
    if $d_i - cur < n$ then
        $i \leftarrow i + 1$;
    else
        $cur \leftarrow d_{i-1}$;
        $S \leftarrow S \cup \{d_{i-1}\}$;
```

Algorithme 1: Greedy algo for the gas stations problem.

## Avoiding the lack of petrol. 1

We need to show that a greedy choice always leads to an optimal solution. Let $X = (x_0, \dots, x_p)$ be an optimal solution and let $Y = (y_0, \dots, y_q)$ the solution given by the greedy algorithm. Since $X$ is optimal: $p \leq q$. Let $k$ be the smallest index such that the two solutions differ in $k$:
$\forall i < k, x_i = y_i$ and $x_k \neq y_k$.
The idea is then to modify the solution $X$ by replacing $x_k$ by $y_k$, the result is still a correct solution (with a little drawing it is easy to see what happens):

- $x_k \leq y_k \leq x_{k+1}$, indeed since $Y$ is the greedy solution $y_k \geq x_k$ since the algo chooses the farthest gas station. Moreover, $y_k < x_{k+1}$, otherwise we could replace $x_k$ and $x_{k+1}$ by $y_k$ and we would obtain a better solution than $X$ which is not possible since $X$ is optimal.
- $y_k - x_{k-1} = y_k - y_{k-1} \leq n$ (because $y_{k-1} = x_{k-1}$),
- $x_{k+1} - y_k < x_{k+1} - x_k \leq n$ (since $x_k \leq y_k \leq x_{k+1}$)

The new solution is also an optimal solution because it contains the same number of gas station.

## Avoiding the lack of petrol. 2

Here, the point is to show that an optimal solution from the starting point $d_0$, is also an optimal solution from another starting point $x_i$ for the subproblem restricted to the remaining stations from this point.
Let $X_1 = (x_1, \dots, x_p)$ be an optimal solution from $d_0$. The first stop $x_1$ is a station $d_i$, $i > 0$. We consider the sequence $X_2 = (x_2, \dots, x_p)$ and we will show that it is an optimal solution for the subproblem with stations $(d_i, \dots, d_m)$ with $d_i$ the new starting point.
By contradiction. Assume $X_2$ is not an optimal solution for $(d_i, \dots, d_m)$, thus there exists a better optimal solution $X' = (x'_2, \dots, x'_q)$, $q < p$. But by adding $x_1$ to $X'$, we obtain a solution $(x_1, x'_2, \dots, x'_q)$ better than $X_1$ which leads to a contradiction.
The subsolution of an optimal solution is thus an optimal solution for a subproblem.
**Note:** we show here that if $X_1$ is an optimal solution, then $X_2$ is an optimal solution for the subproblem, but we do NOT show that if $X_2$ is optimal then $X_1$ is. Be careful to the direction of the implication.

## 2 Coin changing

## Coin changing. 1

The greedy algorithm is simple: first we take as many coins of 50 as possible, then as many of 20, then as many of 10, then as many of 5 and finally we take the remaining as coins of 1.
We want to show that there exists an optimal solution beginning with a greedy choice.

Let $M$ be an optimal solution for $C$. We begin with deducing some properties on the number of coins of each sort in the optimal solution. M contains at most 4 coins of 1, otherwise we could replace the 5 1's by a coin of 5. Similarly, $M$ contains at most one coin of 5, 1 of 10 and two of 20 and an unbounded number of coins of 50; but $M$ cannot contain 2 coins of 20 and 1 of 10 (otherwise we would replace them by a coin of 50).
...

## Coin changing. 1 (continued)

if $C \geq 50$, the algo selects a coin of 50. We need to show that there exists an optimal solution with at least one coin of 50. By contradiction, assume no optimal solution contains any coin of 50, thus any optimal solution has coins of 1, 5, 10 and 20. But, according to the properties found previously, a solution without any coin of 50 could contain at most 2 coins of 20, 1 of 5 and 4 of 1, or 1 of 20, 1 of 10, 1 of 5 and 4 of 1:
$2 * 20 + 5 + 4 * 1 = 49 < 50$ or $20 + 10 + 5 + 4 * 1 = 39 < 50$. It is then impossible to obtain 50 and thus the solution contains at least 1 coin of 50 and the greedy algo makes a correct choice.

We make a similar analysis when $20 \leq C < 50$ to conclude that the algo must take at least one coin of 20. We repeat the same arguments when $C < 20$ to justify that the first choice of the algorithm is always the best. Same strategy for other values of $C$...

## Coin changing. 2

Let $M$ be an optimal solution for an amount $C$. Let $p$ be the first coin selected by the algo. This coin is the coin with the largest value while being less or equal than $C$. We are going to show that $M \backslash \{p\}$ is an optimal solution for the amount $C - p$. By contradiction, assume that $M \backslash \{p\}$ is not optimal for $C - p$. Thus, there exists $M'$ an optimal solution for $C - p$ with less coins than $M \backslash \{p\}$. As a consequence, $M' \cup \{p\}$ is a better solution than $M$ for the amount $C$, which leads to a contradiction. Here the notation $M \backslash \{p\}$ means that we subtract one coin of value $p$ to $M$, and $M' \cup \{p\}$ that we add one coin of value $p$ to $M'$. $\cup$ and $\backslash$ do not correspond to usual operators on sets.

## Coin changing. 3

As said before, sort the coins by decreasing order of value, and following this order, maximize the number of coins for each value. The complexity is in $O(m \log m)$ with $m$ the number of coins (since we must sort them), the core of the procedure is then linear in the number of coins.

## Coin changing. 4

Yes, the previous algorithm works, we can use the same arguments as before to prove the greedy choice property and the optimal substructure property. (An optimal solution can have at most $(c-1)$ coins of value $c^i$, for $i < k$).

## Coin changing. 5

Consider a set of 3 coins of values $\{6, 4, 1\}$. For $C = 8$, the algo outputs $8 = 6 + 1 + 1$, while $8 = 4 + 4$ is a better solution.

## Coin changing. 6

When the set of coins can take any values, we do not know if the greedy algorithm can find the optimal solution in the general case. A solution is to use a dynamic programming approach: let $S = \{a_1, \dots, a_m\}$ be a set of coins and let $z(C, i)$ be the minimal number of coins chosen among the first $i$ coins of $S$ to obtain $C$. We can define the following recursive formulation.

$$z(C,i) = min \begin{cases} z(C, i-1) & \text{if coin i is not used} \\ z(C - a_i, i) + 1 & \text{if coin i is used at least once.} \end{cases}$$

Note that the initialization is done by $z(C, i) = 0$ if $C = 0$, $z(C, i) = \infty$ if $i = 0$ or $C < \infty$. The DP algo uses an external loop on $i$ and an internal one on $C$. The complexity is thus in $O(m \times C)$.
But the complexity depends on the value of one parameter which does not correspond to the standard definition of complexity that depends on the input size. Then, the algorithm is called *pseudo-polynomial*.

# 3 Knapsack problem

## Greedy approaches and the Knapsack problem.

1. Consider the following 3 objects:
   - $o_1$ with $v_1 = 8$ and $w_1 = 10$, $o_2$ with $w_2 = 6$ and $v_2 = 6$ and $o_3$ with $w_3 = 4$ and $v_3 = 7$

   Set $W = 10$, the greedy approach gives $\{o_1\}$ while $\{o_2, o_3\}$ is better.
2. Consider the following 3 objects:
   - $o_1$ with $w_1 = 3$ and $v_1 = 1$, $o_2$ with $w_2 = 4$ and $v_2 = 2$ and $o_3$ with $w_3 = 7$ and $v_3 = 10$

   Set $W = 10$, the greedy approach gives $\{o_1, o_2\}$ while $\{o_1, o_3\}$ is better.
3. Consider the following 3 objects:
   - $o_1$ with $w_1 = 6$ and $v_1 = 10$ with ratio of 1.666, $o_2$ with $w_2 = 5$ and $v_2 = 7$ with ratio of 1.4 and $o_3$ with $w_3 = 5$ and $v_3 = 6$ with ratio of 1.2.

   Set $W = 10$, the greedy approach gives $\{o_1\}$ while $\{o_2, o_3\}$ is better.

## Greedy approaches and the Knapsack problem. 4

```
begin
    Sort the objects according to the ratio $\frac{v_i}{w_i}$ in decreasing order;
    $weight \leftarrow 0$;/*current weight*/ $val \leftarrow 0$;/*current value*/ $i \leftarrow 1$;
    while $weight < W$ do
        Take $o_i$: the next in the list;
        if $W - weight > w_i$ then
            $weight \leftarrow weight + w_i$;
            $val \leftarrow val + v_i$;
            $i \leftarrow i + 1$ //i.e. remove $o_i$ from the list;
        else
            $val \leftarrow val + v_i * (W - weight)/w_i$ /*fraction object */;
            $weight \leftarrow weight + (W - weight)$;
end
```

The complexity is $O(n \log n)$ because we have to sort the object according to the value/weight ratios in decreasing order.

## Greedy approaches and the Knapsack problem. 4

To prove the optimality of the approach we can use the same arguments as in the exercise *lack of petrol* of the exercise sheet. You consider an optimal solution and the one from the greedy algo, then you take the smallest index (ie the first object) where the two solutions differ - the two quantities differ - and in the optimal solution we change the quantity of this object such that it is equal to the greedy one (we also need remove the additional quantity from the quantities of the remaining objects with lower ratio to keep the same weight). This is always possible because of the ordering considered. We conclude as previously.

## Greedy Choice Property

Let $G = \{x_1, \dots, x_n\}$ denotes the greedy solution where $x_i$ corresponds to the fraction of object $o_i$ included in the knapsack and $O = \{y_1, \dots, y_n\}$ an optimal solution with $y_i$ denoting the fraction taken for object $o_i$. The objects are ordered with respect to decreasing value/weight ratio.

Note that from the greedy construction, there must exist a $j$ that is the first index where $x_j \neq 1$, and $x_i = 1$ for all $i < j$ and $x_i = 0$ for all $i > j$.
We assume $\sum_{i=1}^{n} w_i * y_i = M$ (otherwise we can put more objects in the knapsack) and similarly $\sum_{i=1}^{n} w_i * x_i = M$. (Note that if $\sum_{i=1}^{n} w_i < M$ then the solution taking all the objects is optimal, nothing to prove)

Let $k$ be the first index where the two solutions differ, *i.e.* $x_k \neq y_k$. It must be the case that $x_k > y_k$. Indeed, if $k < j$, then $x_k = 1$ and then $x_k > y_k$. If $k \geq j$ and $y_k > x_k$, then $\sum_{i=1}^{n} w_i * y_i > M$ since we already have that $\sum_{i=1}^{j} w_i * x_i = M$, which is not possible.
Then we increase $y_k$ until it becomes $x_k$, while decreasing some or all the $y_i$ for $k + 1 \leq i \leq n$ so that the total weight of the knapsack remains the same. Let $Z = \{z_1, \dots, z_n\}$ be the new solution, where $y_i = z_i$ for $i < k$.

## Greedy Choice Property (ctd)

Observe $w_k * (z_k - y_k) = \sum_{i=k+1}^{n} w_i * (y_i - z_i)$ to maintain feasibility. Then, we have:

$$\begin{aligned}
\sum_{i=1}^{n} w_i * z_i &= \sum_{i=1}^{n} w_i * z_i + \sum_{i=k+1}^{n} w_i * y_i - \sum_{i=k+1}^{n} w_i * y_i \\
&= \sum_{i=1}^{k-1} w_i * y_i + w_k * z_k + w_k * y_k - w_k * y_k + \sum_{i=k+1}^{n} w_i * y_i + \sum_{i=k+1}^{n} w_i(z_i - y_i) \\
&= \sum_{i=1}^{n} w_i * y_i + w_k * (z_k - y_k) - \sum_{i=k+1}^{n} w_i * (y_i - z_i) \\
&= \sum_{i=1}^{n} w_i * y_i + [\sum_{i=k+1}^{n} w_i * (y_i - z_i) - \sum_{i=k+1}^{n} w_i * (y_i - z_i)] \\
&= \sum_{i=1}^{n} w_i * y_i .
\end{aligned}$$

We have shown that the new solution fits in the knapsack.

## Greedy Choice Property (ctd) - value

Observe that $w_k * (z_k - y_k) = \sum_{i=k+1}^{n} w_i * (y_i - z_i)$ to maintain feasibility. Then, we have:

$$\begin{aligned}
\sum_{i=1}^{n} v_i * z_i &= \sum_{i=1}^{k-1} v_i * y_i + v_k z_k + \sum_{i=k+1}^{n} v_i * z_i + \sum_{i=k}^{n} y_i v_i - \sum_{i=k}^{n} y_i v_i \\
&= \sum_{i=1}^{n} v_i * y_i + v_k * (z_k - y_k) - \sum_{i=k+1}^{n} v_i * (y_i - z_i) \\
&= \sum_{i=1}^{n} v_i * y_i + \frac{v_k}{w_k}(z_k - y_k) w_k - \sum_{i=k+1}^{n} \frac{v_i}{w_i} * (y_i - z_i) * w_i \\
&\geq \sum_{i=1}^{n} v_i * y_i + \frac{v_k}{w_k}(z_k - y_k) w_k - \sum_{i=k+1}^{n} \frac{v_k}{w_k} * (y_i - z_i) * w_i \\
&= \sum_{i=1}^{n} v_i * y_i + \frac{v_k}{w_k} \left( (z_k - y_k) w_k - \sum_{i=k+1}^{n} (y_i - z_i) * w_i \right) \\
&= \sum_{i=1}^{n} v_i * y_i. \quad \text{We have shown that the new solution is optimal!}
\end{aligned}$$

($\sum_{i=1}^{n} v_i * z_i > \sum_{i=1}^{n} v_i * y_i$ would be a contraction w.r.t. optimality of $O$ thus the 2 solutions cannot differ in any index, $\sum_{i=1}^{n} v_i * z_i = \sum_{i=1}^{n} v_i * y_i$: the greedy choice for $k$ belongs to an optimal solution, procedure can be repeated for other index)

## Optimal Substructure Property

Consider a fractional knapsack problem with $n$ objects $\{o_1, \ldots, o_n\}$. Let $G = \{x_1, \ldots, x_n\}$, be a (greedy) optimal solution where each $x_i$ denotes the fraction of object $o_i$ used in this solution. For the object $o_1$, the greedy choice takes a fraction $x_1$.
Claim: $G_2 = \{x_2, \ldots, x_n\}$ is an optimal solution for the problem with the object set $S = \{o_2, \ldots, o_n\}$ with weight limit $W - x_1 * w_1$.

**Proof.** (Easy) By contradiction. Suppose there exists another solution $G'$ better than $G_2$: $\sum_{i=2}^n x'_i * v_i > \sum_{i=2}^n x_i * v_i$ with $\sum_{i=2}^n x'_i * w_i \leq W - x_1 * w_1$.
Then, we have that $x_1 * v_1 + \sum_{i=2}^n x'_i * v_i > \sum_{i=1}^n x_i * v_i$ and $x_1 * w_1 + \sum_{i=2}^n x'_i * w_i \leq W$. Thus solution consisting in adding a fraction $x_1$ of $o_1$ in G' is acceptable and is better than the optimal solution $G$.
$\Rightarrow$ Contradiction.

## Dynamic programming and the Knapsack Problem

### Structure of the optimal solution - Definition and 2 cases

$OPT(i, w)$: maximum gain (value) from the subset of items $1, \dots, i$ with limit $w$.

- item $i$ is not in the optimal solution :
  OPT selects the best of $\{1, 2, \dots, i-1\}$ with weight limit $w$
  $OPT(i, w) = OPT(i-1, w)$
- $i$ is selection in the optimal solution $i$ :
  OPT selects the best of $\{1, 2, \dots, i-1\}$ with a new weight limit $W$
  $OPT(i, w) = v_i + OPT(i-1, w-w_i)$

### Recursive formulation

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ 0 & \text{if } w = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ max(\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\}) & \text{otherwise} \end{cases}$$

## Algorithm

```
Input: $n, w_1, \ldots, w_n, v_1, \ldots, v_n$
begin
    for $w = 0$ to $W$ do M[0,w]←0;
    for $i = 1$ to $n$ do M[i,0]←0;
    for $i = 1$ to $n$ do
        for $w = 1$ to $W$ do
            if $w_i > w$ then
                M[i,w]←M[i-1,w]
            else
                $\max(M[i-1, w], v_i + M[i-1, w - w_i])$
end
return M[n,W];
```

Complexity $O(n \times W) \Rightarrow$ complexity *pseudo-polynomial* since it does depend on the input size $(2 * n + 1)$.

$\rightarrow$ Exercise: Apply it on the objects of page 15.

Note, we could also store the choice made at each step of the algo for the final solution.