Laboratório de Programação Avançada 2018/19
Week 5 – Dynamic Programming

· U C ·

Universidade de Coimbra

Knapsack problem

- You planning a move and you want to know which art pieces you should select that maximizes the total value but without exceeding the maximum capacity of the truck.
- This is the knapsack problem where each object i in a set of n objects has a value v_i and a weight w_i. The sum of the weights of the chosen objects must not exceed W (capacity constraint).
- Does it also have optimal substructure?

Sub-problem

- Find the objects taken the first $i \leq n$ objects that maximize the value and satisfy the contraint $W' \leq W$.
- Let S be the optimal set of objects, taken from the first i objects, with total value v and total weight $w \leq W'$.

Optimal substructure

- If S contains the i-th object, then by removing it, we have an optimal solution with objects taken from the first $i - 1$ objects that satisfies the constraint without the weight of that object. (we prove this in the following).
- If S does not contain the i-th object, then we have an optimal solution with objects taken from the first $i - 1$ objects that satisfies constraint W'.

1. Let S be the optimal set of objects, taken from the first i objects, with total value v and total weight $w \leq W'$, and using the i-th object.

2. Then, S without that object, with total value $v - v_i$ and total weight $w - w_i$, is optimal for the first $i - 1$ objects and satisfies constraint $W' - w_i$.

Sketch of the proof (by contradiction)

- (negate 2.) Assume that there exists another set of objects, taken from the first $i - 1$ objects, with total value $v' > v - v_i$ and total weight $w' \leq W' - w_i$.

- (contradict 1.) Then, it also exists a set using the i-th object with total value $v' + v_i > v$ and weight $w' + w_i \leq W'$.

Knapsack

Recursive solution: Choose the n-th object:

1. Either use it and solve sub-problem for $W - w_n$ with the remaining $n - 1$ objects, or
2. Do not use it and solve sub-problem for W with the remaining $n - 1$ objects
3. Choose the maximum value of the two.

A simple recursive solution:

Function $knapsack(i, W)$
  if $i = 0$ **then** $\{$base case - no more objects$\}$
    return 0
  if $w_i > W$ **then**
    return $\underbrace{knapsack(i-1, W)}_{\text{don't take the } i\text{-th object}}$
  else
    return $\max(\underbrace{knapsack(i-1, W)}_{\text{don't take the } i\text{-th object}}, \underbrace{v_i + knapsack(i-1, W-w_i)}_{\text{take the } i\text{-th object}})$

It is an exponential approach. Can we do memoizing?

Knapsack

Top-down dynamic programming:

Function *knapsack*(i, W)
  if $i = 0$ **then** {base case - no more objects}
    return 0
  if $T[i, W] \geq 0$ **then**
    return $T[i, W]$
  if $w_i > W$ **then**
    $T[i, W] = knapsack(i-1, W)$
  else
    $T[i, W] = \max(knapsack(i-1, W), v_i + knapsack(i-1, W - w_i))$
  return $T[i, W]$

Table T stores the optimal value for the first i objects and constraint W.

Knapsack

Bottom-up Dynamic Programming:

Function *knapsack*(n, W)
  for $j = 1$ **to** W **do** {1st base case}
    $T[0, j] = 0$
  for $i = 0$ **to** n **do** {2nd base case}
    $T[i, 0] = 0$
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** W **do**
      if $w_i > j$ **then**
        $T[i, j] = T[i-1, j]$
      else
        $T[i, j] = \max(T[i-1, j], v_i + T[i-1, j-w_i])$
  return $T[n, W]$

Also pseudo-polynomial since its time complexity is $O(nW)$.