Dynamic programming 1

Mauro Dell'Amico
DISMI, Universitá di Modena e Reggio Emilia
mauro.dellamico{at}unimore.it, www.or.unimore.it

Dynamic Programming(DP)

- Originally introduced by Bellman (1957) to solve optimization problems that can be decomposes into *phases* (mainly related to optimal control)
- The whole problem decomposes into a sequence of smaller problems with a **single** decision variable
- DP uses a recursion to optimize one variable at a time, but the design of the recursion and the data stored (*states*) guarantee that the final solution is the optimal one
- Note that not all problems can be solved by using DP. They must have some special structure that allow to decompose them.

Decomposition

Let us consider the following problem, representing a decision process with variables $x_1, \dots, x_T$, on T periods or *phases*:

$$z = \max_{x_1,\dots,x_T} \sum_{t=1}^{T} f^t(s^{t-1}, x_t) \quad (1)$$

$$s^t = \Phi_t(s^{t-1}, x_t) \qquad t = 1, \dots, T \quad (2)$$

$$s^0 \qquad \text{given} \quad (3)$$

At the beginning of each phase t the problem is in state s^{t-1} and the decision x_t transform the problem into state s^t. The initial state s^0 is given.

- The state resume the decisions of the previous periods
- the contribution to the objective function for period t depends **only** on s^{t-1} and x_t

Decomposition

Consider phase 1: we have the problem

$$f^1(s^1) = \max_{x_1} \; f^1(s^0, x_1)$$

Given the optimal choice $\hat{x}_1$ we obtain the state $\hat{s}^1 = \Phi_1(s^0, \hat{x}_1)$
The next problem for $t = 2$ is

$$f^2(s^2) = \max_{x_1,x_2}(f^1(s^0, x_1) + f^2(s^1, x_2))$$

$$s^1 = \Phi_1(s^0, x_1) \quad (4)$$

Note that the optimal value for f^2 is not necessarily that with $\hat{x}_1$

To solve the phase 2 problem we must consider **all** the states s^1!

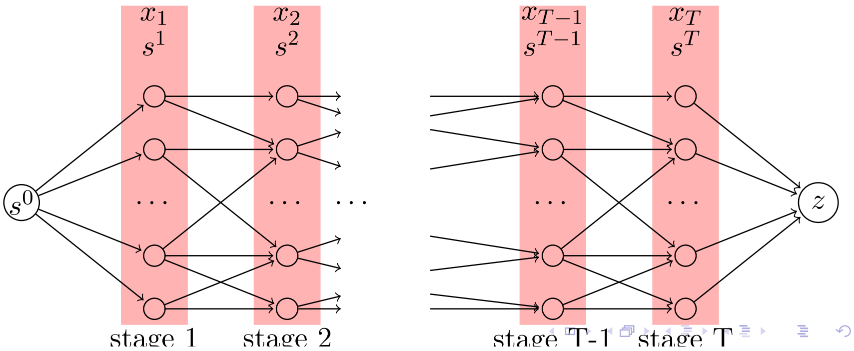
Let $s^1_1 \dots, s^1_{q_1}$ be the states, generated with decisions x_1 by (4)

$$f^2(s^2) = \max_{s^1 \in \{s^1_1 \dots, s^1_{q_1}\}} (f^1(s^1) + \max_{x_2} \; f^2(s^1, x_2))$$

Several states s^2 are generated accordingly to decision x_2.

Resume

1. The problem must be decomposed in **phases** or **stages**. Each phase is associated with **one** decision.
2. In the end of each phase j the problem can be in one of a finite set of **states** $(s^j_1, \dots, s^j_{q_j})$.
3. It is given an objective function $f^j(s^j_i)$ giving the optimal solution value for phase j and final state s^j_i (gives the optimal decisions $x_1, \dots, x_j$ leading to final state s^j_i).



- To implement a DP algoritm we need the *recursion* used at each stage j to compute $f^j(s^j_i)$, for each state s^j_i
- The recursion uses **only** the information on the status (and values) of the previous stage $j-1$
- The pair (f^j, s^j_i) sum up the sequence of optimal decisions needed to reach s^j_i
- The process arrives in a state s^j_i
  - starting from several states s^{j-1}_h;
  - using different decisions x_j

DP does not enumerate all the decision's sequences, since some paths "collapse" into the same state

Knapsack 0-1

- One **phase** for each item j. Decision are *select/do not select*: $x_j \in \{0,1\}$
- The **states** are the possible filling of the bin $0, 1, \ldots, c$
- function $f^j(s)$ is the optimal profit using the first j items and a bin of size s

Ricursion

– if $s < w_j$ item j cannot be inserted : $x_j = 0$

– if $s \geq w_j$:

decision	profit	
j not selected:	$f^{j-1}(s)$	
j selected:	$p_j + f^{j-1}(s - w_j)$	$\leftarrow$ opt. with cap. $s - w_j$

$$f^j(s) = \begin{cases} f^{j-1}(s) & \text{if } s < w_j \\ \max\left(p_j + f^{j-1}(s-w_j), f^{j-1}(s)\right) & \text{if } s \geq w_j \end{cases}$$

$$s = 0, 1, \ldots, c; j = 1, \ldots, n$$

$$f^0(s) = 0, \qquad s = 0, 1, \ldots, c$$

DP-knapsack-01 (versione 1)
input c, (p_j, w_j) for $j = 1, \ldots, n$.
Output selected objects $(J^n(c))$ and profit $(f^n(c))$

```
for s = 0 to c do  f^0(s) = 0, J^0(s) = ∅;
for j = 1 to n do
    //stage j
    for s = 0 to w_j − 1 do f^j(s) = f^{j−1}(s), J^j(s) = J^{j−1}(s);
    for s = w_j to c do
        if p_j + f^{j−1}(s − w_j) > f^{j−1}(s) then
          f^j(s) = f^{j−1}(s − w_j) + p_j,  J^j(s) = J^{j−1}(s − w_j) ∪ {j};
        else
          f^j(s) = f^{j−1}(s),  J^j(s) = J^{j−1}(s);
    end
end
```

Example

$p = (10, 5, 8, 6)$ $w = (2, 3, 4, 3)$ $c = 8$

	0	1	2	3	4	5	6	7	8
f^0	0	0	0	0	0	0	0	0	0
f^1	0	0	10	10	10	10	10	10	10
f^2	0	0	10	10	10	15	15	15	15
f^3	0	0	10	10	10	15	18	18	18
f^4	0	0	10	10	10	16	18	18	21

	0	1	2	3	4	5	6	7	8
J^0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
J^1	$\emptyset$	$\emptyset$	{1}	{1}	{1}	{1}	{1}	{1}	{1}
J^2	$\emptyset$	$\emptyset$	{1}	{1}	{1}	{1,2}	{1,2}	{1,2}	{1,2}
J^3	$\emptyset$	$\emptyset$	{1}	{1}	{1}	{1,2}	{1,3}	{1,3}	{1,3}
J^4	$\emptyset$	$\emptyset$	{1}	{1}	{1}	{1,4}	{1,3}	{1,3}	{1,2,4}

Complexity

for $j = 1$ **to** n
    for $s = 0$ **to** $w_j - 1$ **do** $f^j(s) = f^{j-1}(s), J^j(s) = J^{j-1}(s)$
    for $s = w_j$ **to** c **do**
        if $p_j + f^{j-1}(s - w_j) > f^{j-1}(s)$
          $f^j(s) = f^{j-1}(s - w_j) + p_j,\ J^j(s) = J^{j-1}(s - w_j) \cup \{j\};$
        else
          $f^j(s) = f^{j-1}(s),\ J^j(s) = J^{j-1}(s);$
        endif
    end for
endfor

The "core" of the algorithm has en external loop on j running for n times. For each j there is a loop capacity s iteratiog $O(c)$ times. Inside this loop we have fixed number of operations. Assuming the assignment of a set J can be done on $O(1)$ the overall time complexity is $O(nc)$ $\Leftarrow$ pseudopolynomial