Dynamic programming 2

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

DP-knapsack-01

In a knapsack problem we are given n items with weights $w_1, \dots, w_n$, profits $p_1, \dots, p_n$ and a bin of capacity c. We want to find the subset of items that maximizes the profit while not exceeding the in capacity.

- We have presented and $O(nc)$ Dynamic Programming algorithm to solve the problem, where each stage has $c+1$ states given by the possible filling of the bin
- We can design a second DP algorithm by associating the states to the possible profits: $0, \dots, \sum_{j=1}^{n} p_j$

Another DP algorithm for knapsack 0-1

- In this method we assume that the **states** are given by the **profits** : $0, 1, \dots, P = \sum_{j=1}^{n} p_j$
- The function $f^j(r)$ gives the optimal **filling** using the first j items and giving a profit equal to r.

Recursion

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

$$r = 0, 1, \dots, P, \quad j = 1, \dots, n$$

$$f^0(0) = 0, \quad f^0(r) = +\infty, \qquad r = 1, \dots, P$$

DP-knapsack-01-Profits
input c, (p_j, w_j) for $j = 1, \dots, n$
output Optimal profit r^* and corresponding items $J^n(r^*)$

Compute $P = \sum_{j=1}^{n} p_j$
for $r = 0$ **to** P **do** $f^0(r) = +\infty, J^0(r) = \emptyset$;
$f^0(0) = 0$;
for $j = 1$ **to** n **do**
  for $r = 0$ **to** $p_j - 1$ **do** $f^j(r) = f^{j-1}(r); J^j(r) = J^{j-1}(r)$
  for $r = p_j$ **to** P
    if $w_j + f^{j-1}(r - p_j) < f^{j-1}(r)$ **then**
      $f^j(r) = f^{j-1}(r - p_j) + w_j, \; J^j(r) = J^{j-1}(r - p_j) \cup \{j\}$;
    else
      $f^j(r) = f^{j-1}(r), \; J^j(r) = J^{j-1}(r)$;
    endif
  endfor
endfor
Compute $r^* = \max\{r : f^n(r) \leq c\}$

Complexity $O(n \sum_{j=1}^{n} p_j)$: to be used if $\sum_{j=1}^{n} p_j < c$

Shortest path

Shortest path

Given a digraph $G = (V, A)$ with arc costs c_{ij}, find the shortest path from a given vertex $s \in V$ to all other vertices.

- We can decompose the problen into $n - 1$ phases, by associating to phase k the paths using **at most** k arcs
- Phase k computes the shortest paths with **at most** k arcs, by using the results of phase $k - 1$ where we have all the optimal paths using $k - 1$ arcs

The recursion is

$$f^k(j) = \min\left(f^{k-1}(j), \min_{(i,j)\in A}(f^{k-1}(i) + c_{ij})\right)$$

with $f^0(s) = 0,\ f^0(j) = \infty,\ j \in V \setminus \{s\}$

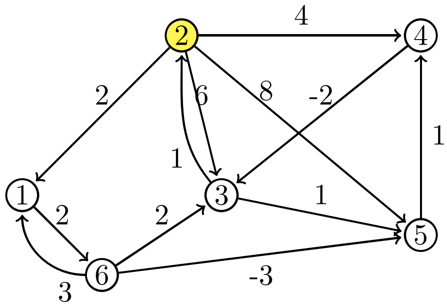
Bellman-Ford shortest path
input $G = (V, A)$ c_{ij}, $(i, j) \in A$.
output cost of the shortest path $f^{n-1}(j)$ $j \in V$

```
forall j ∈ V \ {s} do f(j) = +∞, pred_j = ∅;
f(s) = 0, pred_s = s;
for k = 1 to n − 1  do // phases
    forall j ∈ V \ {s} do // states
        f^k(j) = f^{k−1}(j);
        forall (i, j) ∈ A  do
            if f^{k−1}(i) + c_ij < f^k(j)
              f^k(j) = f^{k−1}(i) + c_ij; pred_j = i;
            endif
        endfor
    endfor
endfor
//Checks for negative cycles
forall (i, j) ∈ A
    if f^{n−1}(i) + c_ij < f^{n−1}(j) return "negative cycle"
endfor
```

Bellman-Ford complexity

```
fork = 1 to n − 1  do // phases
    forall j ∈ V \ {s} do // states
        f^k(j) = f^{k−1}(j);
        forall (i, j) ∈ A  do
            if f^{k−1}(i) + c_ij < f^k(j)
              f^k(j) = f^{k−1}(i) + c_ij; pred_j = i;
            endif
        endfor
    endfor
endfor
```

The first two loops: **for** and **forall** run in $O(|V|)$ time.
The inner loop **for all** loops scan all the arcs,
so the overall complexity is $O(|V|^2|A|) \leq O(|V|^3)$



Source node = 2

Iter	$f(j)$						pred					
	1	2	3	4	5	6	1	2	3	4	5	6
0	-	0	-	-	-	-	-	2	-	-	-	-
1	2	0	6	4	8	-	2	2	2	2	2	-
2	2	0	2	4	7	4	2	2	4	2	3	1
3	2	0	2	4	1	4	2	2	4	2	6	1
4	2	0	2	2	1	4	2	2	4	5	6	1
5	2	0	0	2	1	4	2	2	4	5	6	1