Dynamic Programming

Design and Analysis of Algorithms
Andrei Bulatov

Knapsack

The Knapsack Problem

Instance:

A set of n objects, each of which has a positive integer value v_i and a positive integer weight w_i. A weight limit W.

Objective:

Select objects so that their total weight does not exceed W, and they have maximal total value

Idea

A simple question: Should we include the last object into selection?

Let OPT(n,W) denote the maximal value of a selection of objects out of {1, …, n} such that the total weight of the selection doesn't exceed W

More general, OPT(i,U) denote the maximal value of a selection of objects out of {1, …, i} such that the total weight of the selection doesn't exceed U

Then

OPT(n,W) = max{ OPT(n – 1, W), OPT(n – 1, W – w_i) + v_i}

Algorithm (First Try)

Knapsack(n,W)

```
set V1:=Knapsack(n-1,W)
set V2:=Knapsack(n-1,W- w_i )
output max(V1,V2+v_i )
```

Is it good enough?

Example

Let the values be 1,3,4,2, the weights 1,1,3,2, and W = 5

Recursion tree

Another Idea: Memoization

Let us store values OPT(i,U) as we find them

We need to store (and compute) at most $n \times W$ numbers

We'll do it in a regular way:

Instead of recursion, we will compute those values starting from smaller ones, and fill up a table

Algorithm (Second Try)

Knapsack(n,W)

```
array  M[0..n,0..W]
set M[0,w]:=0 for each w=0,1,...,W
for i=1 to n do
   for w=0 to W do
      set M[i,w]:= max{M[i-1,w],M[n-1,w-w_i]+v_i}
   endfor
endfor
```

Example

Example

Let the values be 1,3,4,2, the weights 1,1,3,2, and W = 5

w \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	1	3	3	3
2	0	1	4	4	4
3	0	1	4	4	5
4	0	1	4	7	7
5	0	1	4	8	8

M[i,w] = max{ M[i – 1, w], M[n – 1,w – w_i] + v_i }

Shortest Path

Suppose that every arc e of a digraph G has length (or cost, or weight, or …) len(e)

But now we allow negative lengths (weights)

Then we can naturally define the length of a directed path in G, and the distance between any two nodes

The s-t-Shortest Path Problem

Instance:

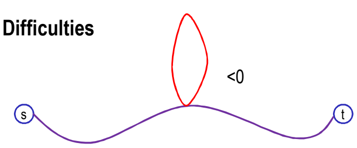
Digraph G with lengths of arcs, and nodes s,t

Objective:

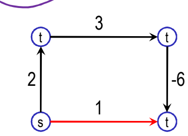
Find a shortest path between s and t

Shortest Path: Difficulties

Negative Cycles.



Greediness fails



Adding constant weight to all arcs fails

Shortest Path: Observations

Assumption

There are no negative cycles

Lemma

If graph G has no negative cycles, then there is a shortest path from s to t that is simple (i.e. does not repeat nodes), and hence has at most n – 1 arcs

Proof

If a shortest path P from s to t repeat a node v, then it also include a cycle C starting and ending at v.

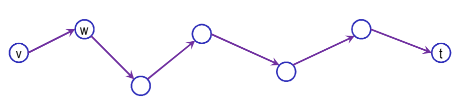
The weight of the cycle is non-negative, therefore removing the cycle makes the path shorter (no longer).

QED

Shortest Path: Dynamic Programming

We will be looking for a shortest path with increasing number of arcs

Let OPT(i,v) denote the minimum weight of a path from v to t using at most i arcs



Shortest v – t path can use i – 1 arcs. Then OPT(i,v) = OPT(i – 1,v)

Or it can use i arcs and the first arc is vw. Then

OPT(i,v) = len(vw) + OPT(i – 1,w)

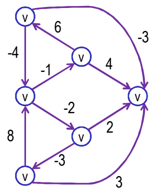
$$OPT(i,v) = \min\{OPT(i-1,v), \min_{w \in V}\{OPT(i-1,w) + len(vw)\}\}$$

Shortest Path: Algorithm

Shortest-Path(G,s,t)

```
set n:=|V|          /*number of nodes in G
array  M[0..n-1,V]
set M[0,t]:=0 and M[0,v]:=∞ for each v∈V-{t}
for i=1 to n-1 do
   for v∈V do
    set M[i,w]:=min{M[i-1,v],min_{w∈V}{M[i-1,w]+len(vw)}}
   endfor
endfor
return m[n-1,s]
```

Example



	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

M[i,w] = min{ M[i – 1, v], $\min_{w \in V}$ { M[i – 1, w] + len(vw) }}