Lecture 14: March 7, 2013

Today, we will analyze the amount of "computational effort" of solving the knapsack problem using DP.

1 Continued from last time: The knapsack problem

Last lecture, we consider the following example of the knapsack problem, which we also see in Recitation 7A. We have a knapsack of weight capacity $W = 13$ pounds and $N = 4$ products, each is available in unlimited quantities.

Product (i)	Weight per unit (w_i)	Value per unit (d_i)
1	5	9
2	3	4
3	2	3
4	1	0.5

A DP formulation that we saw:

1. Stages: The stages correspond to a product type. That is, at stage k, we consider products 1 through k, making a decision on the quantity of product k to take.

2. States: At stage k, we consider various values for the state $\widehat{W}$, which corresponds to a leftover weight capacity of $\widehat{W}$. In particular, we consider $\widehat{W} = 0, 1, \ldots, W$ (here, $W = 13$).

3. Decisions: At stage k and state $\widehat{W}$, we decide how many units of product k we should take, given that we have a leftover capacity of $\widehat{W}$ for products 1 through k.

   So, if x_k denote the number of units of product k to take, then if the leftover capacity is $\widehat{W}$, the allowable values of x_k are:

   $$0, 1, \ldots, \left\lfloor \frac{\widehat{W}}{w_k} \right\rfloor,$$

   where $\left\lfloor \frac{\widehat{W}}{w_k} \right\rfloor$ is the maximum number of units of product k that fits into a knapsack of leftover capacity $\widehat{W}$.

4. Optimization function: we let $f_k^*(\widehat{W})$ denote the maximum value of a knapsack with leftover weight capacity $\widehat{W}$ when we only consider products 1 through k.

5. According to (4) above, $f_k^*(\widehat{W})$ is easiest to compute at stage $k = 1$:

   $$f_1^*(\widehat{W}) = \text{the maximum value of a knapsack with leftover weight capacity } \widehat{W} \text{ when we only consider product 1.}$$

   So,

   $$f_k^*(\widehat{W}) = \left\lfloor \frac{\widehat{W}}{w_1} \right\rfloor \cdot d_1; \quad x_1^* = \left\lfloor \frac{\widehat{W}}{w_1} \right\rfloor,$$

   for all values of $\widehat{W} = 0, 1, \ldots, W$ (here, $W = 13$).

6. Recurrence relation:

$$\begin{aligned} f_k^*(\widehat{W}) &= \text{the maximum value of a knapsack with leftover weight capacity } \widehat{W} \\ &\quad \text{when we only consider products 1 through } k \\ &= \max_{\substack{\text{allowable} \\ \text{values of } x_k}} \left\{ \text{value from } x_k \text{ units of prod } k + \left(\begin{array}{c} \text{max value of a knapsack with} \\ \text{leftover weight capacity of } \widehat{W} - x_k w_k \\ \text{for products 1 through } k-1 \end{array} \right) \right\} \\ &= \max_{x_k = 0, \ldots, \lfloor \widehat{W}/w_k \rfloor} \left\{ x_k \cdot d_k + f_{k-1}^*(\widehat{W} - x_k w_k) \right\} \end{aligned}$$

7. Computation: We complete the following DP table (i.e. we solve for $f_k^*(\widehat{W})$ for each $k = 1, \ldots, N$ and each $\widehat{W} = 0, \ldots, W$) starting from the column for stage 1, which provides our boundary conditions, then the column for stage 2, etc.

$\widehat{W}$	$f_1^*(\widehat{W}),\ x_1^*$	$f_2^*(\widehat{W}),\ x_2^*$	$f_3^*(\widehat{W}),\ x_3^*$	$f_4^*(\widehat{W}),\ x_4^*$
0				
1				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
12				
13				

8. The optimal value of the overall problem is given by $f_N^*(W)$, which in our case is $f_4^*(13)$. We trace back to obtain the optimal solution.

Remark. How much work does it take to solve the knapsack problem using the above DP approach?

- In our example, our state space has size 14 and there are $14 \times x4 = 56$ cells to fill.
- In general, the state space has size $W + 1$ and there are $(W + 1)N$ cells to fill.
- So, we say that the state space is of order $O(W)$ and the number of cells is of order $O(WN)$. [See last section on this "Big-Oh" notation.]
- (To compute each cell in the kth stage, we consider the maximum among up to $\lfloor \widehat{W}/w_k \rfloor + 1$ possible values of x_k. Assuming that each w_k is a positive integer, the number of possible values of each x_k is at most $\widehat{W} + 1 = O(W)$. So in total, our computational effort is around $O(NW^2)$.)

2 The knapsack problem with multiple capacity constraints

Suppose we now have a knapsack that has a volume capacity constraint in addition to a weight capacity constraint. Consider the previous example: Suppose that the weight capacity is still $W = 13$ pounds, but it now also has a volume capacity of $V = 15$ liters. The volumes of our $N = 4$ products are given in the following table:

Product (i)	Weight per unit (w_i)	Volume per unit (v_i)	Value per unit (d_i)
1	5	4	9
2	3	5	4
3	2	3	3
4	1	1	0.5

Then, besides keeping track of leftover weight capacity (as our "state"), we also need to keep track of leftover volume capacity, as a "state". Hence, the DP approach becomes:

1. Stages: The stages correspond to a product type. That is, at stage k, we consider products 1 through k, making a decision on the quantity of product k to take.

2. States: At stage k, we consider various values for the state $(\widehat{W}, \widehat{V})$, which corresponds to a leftover weight capacity of $\widehat{W}$ and a leftover volume capacity $\widehat{V}$. Note that our state is now a 2-dimensional vector.

   In particular, at each stage k, we have to consider all possible pairs of values $\widehat{W} = 0, 1, \ldots, W$ (here, $W = 13$) and $\widehat{V} = 0, 1, \ldots, V$ (here, $V = 15$).

3. Decisions: At stage k and state $(\widehat{W}, \widehat{V})$, we decide how many units of product k we should take, given that we have a leftover weight and volume capacities of $(\widehat{W}, \widehat{V})$ for products 1 through k.

   So, if x_k denote the number of units of product k to take, then if the leftover capacity is $\widehat{W}$, the allowable values of x_k are:

   $$0, 1, \ldots, \min\left(\left\lfloor \frac{\widehat{W}}{w_k} \right\rfloor, \left\lfloor \frac{\widehat{V}}{v_k} \right\rfloor\right),$$

   that is, x_k has to be less than or equal to both $\left\lfloor \frac{\widehat{W}}{w_k} \right\rfloor$ and $\left\lfloor \frac{\widehat{V}}{v_k} \right\rfloor$.

4. Optimization function: we let $f_k^*((\widehat{W}, \widehat{V}))$ denote the maximum value of a knapsack with leftover weight and volume capacities $(\widehat{W}, \widehat{V})$ when we only consider products 1 through k.

5. According to (4) above, $f_k^*((\widehat{W}, \widehat{V}))$ is easiest to compute at stage $k = 1$:

   $f_1^*((\widehat{W}, \widehat{V}))$ = the maximum value of a knapsack with leftover weight capacity $\widehat{W}$ when we only consider product 1.

   So,

   $$f_k^*(\widehat{W}, \widehat{V}) = x_1^* \cdot d_1; \quad x_1^* = \min\left(\left\lfloor \frac{\widehat{W}}{w_1} \right\rfloor, \left\lfloor \frac{\widehat{V}}{v_1} \right\rfloor\right),$$

   for all values of $\widehat{W} = 0, 1, \ldots, W$ and $\widehat{V} = 0, 1, \ldots, V$ (here $W = 13, V - 15$).

6. Recurrence relation:

$$\begin{aligned} f_k^*(\widehat{W},\widehat{V}) &= \text{the maximum value of a knapsack with leftover weight capacity } \widehat{W} \\ &\quad \text{when we only consider products 1 through } k \\ &= \max_{\substack{\text{allowable} \\ \text{values of } x_k}} \left\{ \text{value from } x_k \text{ units of prod } k + \left(\begin{array}{c} \text{max value of a knapsack with leftover} \\ \text{capacities of } (\widehat{W}-x_k w_k, \widehat{V}-x_k v_k) \\ \text{for products 1 through } k-1 \end{array} \right) \right\} \\ &= \max_{x_k=0,\ldots,\min(\lfloor \widehat{W}/w_k \rfloor, \lfloor \widehat{V}/v_k \rfloor,)} \left\{ x_k \cdot d_k + f_{k-1}^*(\widehat{W}-x_k w_k, \widehat{V}-x_k v_k) \right\} \end{aligned}$$

7. Computation: We complete the following DP table (i.e. we solve for $f_k^*(\widehat{W},\widehat{V})$ for each $k = 1,\ldots,N$ and each $\widehat{W} = 0,\ldots,W$) starting from the column for stage 1, which provides our boundary conditions, then the column for stage 2, etc.

$(\widehat{W},\widehat{V})$	$f_1^*(\widehat{W}),\ x_1^*$	$f_2^*(\widehat{W}),\ x_2^*$	$f_3^*(\widehat{W}),\ x_3^*$	$f_4^*(\widehat{W}),\ x_4^*$
$(0,0)$				
$(0,1)$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(0,14)$				
$(0,15)$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(13,14)$				
$(13,15)$				

or, rearranging the table: for each stage k, we fill out:

	$\widehat{V}=0$	$\widehat{V}=1$	$\cdots$	$\widehat{V}=15$
$\widehat{W}=0$	$f_k^*(0,0)=...,\ x_k^*=...$	$f_k^*(0,1)=...,\ x_k^*=...$	$\cdots$	$f_k^*(0,15)=...,\ x_k^*=...$
$\widehat{W}=1$	$f_k^*(1,0)=...,\ x_k^*=...$	$f_k^*(1,1)=...,\ x_k^*=...$	$\cdots$	$f_k^*(1,15)=...,\ x_k^*=...$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\widehat{W}=13$	$f_k^*(13,0)=...,\ x_k^*=...$	$f_k^*(13,1)=...,\ x_k^*=...$	$\cdots$	$f_k^*(13,15)=...,\ x_k^*=...$

8. The optimal value of the overall problem is given by $f_N^*(W,V)$, which in our case is $f_4^*(13,15)$. We trace back to obtain the optimal solution.

Remark. How much work does it take to solve the knapsack problem using the above DP approach?

- In our example, the state space has size 14×15 and there are $14 \times 15 \times 4 = 896$ cells to fill.
- In general, the state space has size $(W+1)(V+1)$ and there are $(W+1)(V+1)N$ cells to fill.
- So, we say that the state space is of order $O(WV)$ and the number of cells is of order $O(NWV)$.
- (To compute each cell in the kth stage, we consider the maximum among up to $\lfloor \widehat{W}/w_k \rfloor + 1$ possible values of x_k. Assuming that each w_k, v_k is a positive integer, the number of possible values of each x_k is at most $\min(\widehat{W}+1, \widehat{V}+1) = O(W)$. So in total, our computational effort is around $O(NVW^2)$.)

Comparison between single-constraint and multiple-constraint knapsack:

- With just one capacity constraint (just weight capacity), our state spae is 1-dimensional with size $O(W)$, and we have $O(NW)$ cells to fill.
- With two capacity constraints (weight and volume capacities), our state-space is 2-dimensional, with size $O(WV)$ and we have $O(NWV)$ cells to fill.
- Suppose that we have d capacity constraints and for each constraint, the capacity is of order $O(C)$. Then then our state space is d-dimensional, the size of the state space is $O(C^d)$ and we have $O(NC^d)$ cells to fill.
- This illustrates that as the number of dimensions, d, increase, the amount of computation that we have to do increases exponentially (because d appears as an exponent)
- This issue with DP is often referred to as the "curse of dimensionality" indicating that as the dimension of the state space grow, the amount of computation grows exponentially such that computation becomes intractable (this mean, cannot be done in any reasonable amount of time) very quickly.

3 Another DP formulation of the knapsack problem

As you might have chosen to do in homework, or as you have seen in Recitation 7A, there is another formulation of knapsack:

- Stage $\widehat{W}$ corresponds to remaining weight capacity of $\widehat{W}$; the stages go from $\widehat{W} = 0$ to $\widehat{W} = W = 13$.
- At each stage $\widehat{W}$, we only consider one state: $\widehat{W}$.

  (Note: The act of specifying "Stages" and "states" is part of a way of formalizing our DP approach. In our case here, the "state" is redundant; we don't actually need a state information in this formulation, and could be "skipped". However, here, we decided instead of not having any states, we have a state that is exactly the same as our stage.)
- At stage $\widehat{W}$ and state $\widehat{W}$, we decide whether, given a leftover capacity of $\widehat{W}$, one unit of which product $k = 1, 2, \ldots, N$ we should add into our knapsack. So, the allowable decisions are products k that satisfy
$$w_k \leq \widehat{W},$$
  i.e., any of the products that can fit.
- At stage $\widehat{W}$ and state $\widehat{W}$, our optimization function is $f^*(\widehat{W}) = f^*_{\widehat{W}}(\widehat{W})$ = the maximum value of a knapsack with leftover capacity $\widehat{W}$.
- According to (4) above, the easiest stage $\widehat{W}$ in which to compute $f^*(\widehat{W})$ is when $\widehat{W} = 0$:
$$f^*(\widehat{W}) = \text{the maximum value of a knapsack with leftover capacity } 0 \ = 0, \quad x^*\widehat{W} = \text{none}.$$
- The recurrence relation:
$$\begin{aligned} f(\widehat{W}) &= \text{the maximum value of a knapsack with leftover capacity } \widehat{W} \\ &= \min_{\substack{\text{product } k \\ \text{that fits}}} \left\{ \text{value of product } k + \begin{array}{c}\text{the maximum value of a knapsack} \\ \text{with leftover capacity } \widehat{W}\end{array} \right\} \\ &= \min_{\substack{k=1,\ldots N \\ s.t. w_k \leq \widehat{W}}} \left\{ v_k + f(\widehat{W} - w_k) \right\} \end{aligned}$$
- Computation: complete the following DP table, starting from $\widehat{W} = 0$ to $\widehat{W} = 13$.

$\widehat{W}$	$f^*(\widehat{W}),\ x^*_{\widehat{W}}$
0	
1	
⋮	⋮
12	
13	

- The optimal value of the overall problem is given by $f^*(W)$ which in our case is $f^*(13)$. We trace back to find out the optimal number of units of each product.

Remark.

In our example, there are 14 cells to fill.

In general, there are $(W+1)$ cells to fill. That is, $O(W)$ cells to fill. Note that this is fewer than in the previous DP formulation of knapsack by a factor of N.

(To compute the cell in the $\widehat{W}$th stage, we consider up to N choices of products to be added into the knapsack. So, in total, our computational "effort" is of order $O(NW)$.)

We can also obtain a similar formulation for the "multiple-constraints" knapsack problem. We consider the weight and volume capacity constraints example again. A second formulation:

1. Stage $(\widehat{W}, \widehat{V})$ corresponds to remaining weight and volume capacities of $(\widehat{W}, \widehat{V})$; the stages go from $(\widehat{W}, \widehat{V}) = (0, 0)$ to $(\widehat{W}, \widehat{V}) = (W, V)(= (13, 15))$.

2. At each stage $(\widehat{W}, \widehat{V})$, we only consider one state: $(\widehat{W}, \widehat{V})$.

   (Note: The act of specifying "Stages" and "states" is part of a way of formalizing our DP approach. In our case here, the "state" is redundant; we don't actually need a state information in this formulation, and could be "skipped". We can also use one of them as a state and the other as a stage, etc.)

3. At stage $(\widehat{W}, \widehat{V})$ and state $(\widehat{W}, \widehat{V})$, we decide whether, given a leftover weight and volume capacities pair of $(\widehat{W}, \widehat{V})$, one unit of which product $k = 1, 2, \ldots, N$ we should add into our knapsack. So, the allowable decisions are products k that satisfy

   $$w_k \leq \widehat{W},$$

   and

   $$v_k \leq \widehat{V},$$

   i.e., any of the products that can fit (which means that both capacity constraints are satisfied).

4. At stage $(\widehat{W}, \widehat{V})$ and state $(\widehat{W}, \widehat{V})$, our optimization function is $f^*(\widehat{W}, \widehat{V}) = f^*_{(\widehat{W}, \widehat{V})}(\widehat{W}, \widehat{V}) =$ the maximum value of a knapsack with leftover capacities $(\widehat{W}, \widehat{V})$.

5. According to (4) above, the easiest stage $\widehat{W}$ in which to compute $f^*(\widehat{W}, \widehat{V})$ is when $\widehat{W} = 0$ or $\widehat{V} = 0$:

   $f^*(\widehat{W}, 0) =$ the maximum value of a knapsack with leftover capacity $(\widehat{W}, 0) \ = 0, \ x^*(\widehat{W}, 0) =$ none,

   for any value of $\widehat{W} = 0, 1, \ldots, 13$, and:

   $f^*(0, \widehat{V}) =$ the maximum value of a knapsack with leftover capacity $(0, \widehat{V}) \ = 0, \ x^*(0, \widehat{V}) =$ none,

   for any value of $\widehat{V} = 0, 1, \ldots, 15$.

6. The recurrence relation:

$$\begin{aligned} f(\widehat{W}, \widehat{V}) &= \text{the maximum value of a knapsack with leftover capacity } (\widehat{W}, \widehat{V}) \\ &= \min_{\substack{\text{product } k \\ \text{that fits}}} \left\{ \text{value of product } k + \begin{array}{c}\text{the maximum value of a knapsack with} \\ \text{leftover capacity } (\widehat{W}, \widehat{V})\end{array} \right\} \\ &= \min_{\substack{k = 1, \ldots N \\ s.t. w_k \leq \widehat{W}, v_k \leq \widehat{V}}} \left\{ v_k + f(\widehat{W} - w_k, \widehat{V} - v_k) \right\} \end{aligned}$$

7. Computation: complete the following DP table, starting from the boundary conditions above, to $(\widehat{W}, \widehat{V}) = (13, 15)$:

	$\widehat{V} = 0$	$\widehat{V} = 1$	$\cdots$	$\widehat{V} = 15$
$\widehat{W} = 0$	$f^*(0,0) = ...,\ x^*_{(0,0)} = ...$	$f^*(0,1) = ...,\ x^*_{(0,1)} = ...$	$\cdots$	$f^*(0,15) = ...,\ x^*_{(0,15)} = ...$
$\widehat{W} = 1$	$f^*(1,0) = ...,\ x^*_{(1,0)} = ...$	$f^*(1,1) = ...,\ x^*_{(1,1)} = ...$	$\cdots$	$f^*(1,15) = ...,\ x^*_{(2,15)} = ...$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\widehat{W} = 13$	$f^*(13,0) = ...,\ x^*_{(13,0)} = ...$	$f^*(13,1) = ...,\ x^*_{(13,1)} = ...$	$\cdots$	$f^*(13,15) = ...,\ x^*_{(13,15)} = ...$

   Note that in this case there are a few options on how we can fill out the table after the boundary conditions:

   - First fill all cells where $\widehat{W} = 1$, then all cells where $\widehat{W} = 2$, etc., or
   - First fill all cells where $\widehat{V} = 1$, then all cells where $\widehat{V} = 2$, etc., or
   - First fill all cells where $\widehat{W} + \widehat{V} = 1$, then all cells where $\widehat{W} + \widehat{V} = 2$, etc.

8. The optimal value of the overall problem is given by $f^*(W, V)$ which in our case is $f^*(13, 15)$. We trace back to find out the optimal number of units of each product.

Remark.

In our example, there are 14×16 cells to fill.

In general, there are $(W + 1)(V + 1)$ cells to fill. That is, $O(WV)$ cells to fill. Note that this is fewer than in the previous DP formulation of knapsack by a factor of N.

(To compute the cell in the $(\widehat{W}, \widehat{V})$th stage, we consider up to N choices of products to be added into the knapsack. So, in total, our computational "effort" is of order $O(NWV)$.)

As before, focusing on the number of cells to fill:

- With just one capacity constraint (just weight capacity), our state spae is 1-dimensional, and we have $O(W)$ cells to fill.
- With two capacity constraints (weight and volume capacities), our state-space is 2-dimensional and we have $O(WV)$ cells to fill.
- Suppose that we have d capacity constraints and for each constraint, the capacity is C. Then then our state space is d-dimensional and we have $O(C^d)$ cells to fill
- This illustrates that as the number of dimensions, d, increase, the amount of computation that we have to do increases exponentially (because d appears as an exponent)
- This issue with DP is often referred to as the "curse of dimensionality" indicating that as the dimension of the state space grow, the amount of computation grows exponentially such that computation becomes intractable (this mean, cannot be done in any reasonable amount of time) very quickly.

4 The nonlinear knapsack problem

After the previous sections, you noticed that the second knapsack formulation is "computationally less expensive" than the first formulation, since there are fewer cells to fill (for knapsack with juts a weight capacity constraint of W, the number of cells to fill is of order $O(NW)$ in the first and $O(W)$ in the second; the computational effort is $O(NW^2)$ in the first and $O(NW)$ in the second).

So, why would one ever want to use the first formulation? The following is a version of the knapsack problem, in which you have to use the first kind of formulation:

Consider a knapsack problem with a weight capacity of $W = 13$ pounds. There are 4 types of products, where each product is available in at most 3 units. The weight of each unit of products is as follows:

Product (k)	Weight per unit (w_k)
1	5
2	3
3	2
4	1

The weights are additive. That is, if we decide to take 2 units of product 1, then the total weight from product 1 is $x_1 \times w_1 = 2 \times 5 = 10$ pounds.

However, the values are not additive. The following table has the value if we take i units of product k, for each $i = 1, 2, 3$ and each $k = 1, 2, 3, 4$:

Product (k)	Value of i units of product k (v_{ki})		
	$i = 1$	$i = 2$	$i = 3$
1	9	15	20
2	4	8	12
3	3	4	5
4	0.5	2.5	5

Note that one unit of product 1 has value 9, but two units of product 1 has value $15 \neq x_1 \times 9 = 2 \times 9$, etc.

Question. To think about: why can't we formulate this problem using the second kind of DP formulation? (Hint: try to formulate this problem using the second formulation. In this formulation, certain information is not captured.)

5 The "Big-Oh" Notation