# Discrete Optimization

The Knapsack Problem: Part I

## Goals of the Lecture

Introduce some basic concepts
 Introduce dynamic programming

# The (1-Dimensional) Knapsack Problem

### ► Given a set of items I, each item i ∈ I characterized by

- its weight w<sub>i</sub>
- its value v<sub>i</sub>

# The (1-Dimensional) Knapsack Problem

### ► Given a set of items I, each item i ∈ I characterized by

- its weight w<sub>i</sub>
- its value v<sub>i</sub>
- and
  - a capacity K for a knapsack

# The (1-Dimensional) Knapsack Problem

### ► Given a set of items I, each item i ∈ I characterized by

- its weight w<sub>i</sub>
- its value v<sub>i</sub>

and

- a capacity K for a knapsack
- find the subset of items in I
  - that has maximum value
  - does not exceed the capacity K of the knapsack

### How to model an optimization problem

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into
- express the problem constraints in terms of these variables

they specify what the solutions to the problem are

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into
- express the problem constraints in terms of these variables
  - they specify what the solutions to the problem are
- express the objective function
  - the objective function specifies the quality of each solution

erested into n terms of these

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into
- express the problem constraints in terms of these variables
  - they specify what the solutions to the problem are
- express the objective function
  - the objective function specifies the quality of each solution
- The result is an optimization model

erested into n terms of these

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into
- express the problem constraints in terms of these variables
  - they specify what the solutions to the problem are
- express the objective function
  - the objective function specifies the quality of each solution

### The result is an optimization model

- It is a declarative formulation
  - specify the "what", not the "how"

### How to model an optimization problem

- choose some decision variables
  - they typically encode the result we are interested into
- express the problem constraints in terms of these variables
  - they specify what the solutions to the problem are
- express the objective function
  - the objective function specifies the quality of each solution

### The result is an optimization model

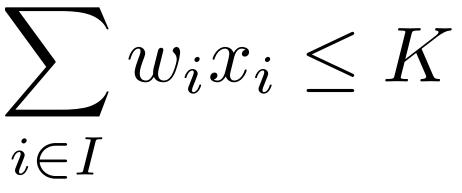
- It is a declarative formulation
  - specify the "what", not the "how"
- There may be many ways to model an optimization problem

### Decision variables

- x<sub>i</sub> denotes whether item i is selected in the solution
  - $x_i = 1$  means the item is selected
  - $x_i = 0$  means that it is not selected

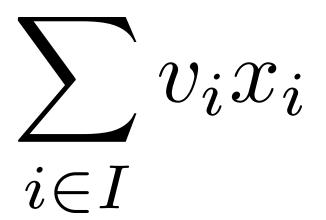
### Decision variables

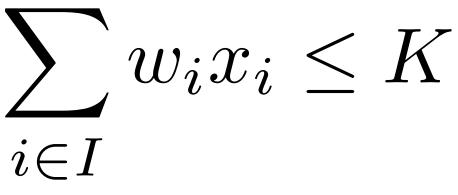
- x<sub>i</sub> denotes whether item i is selected in the solution
  - $x_i = 1$  means the item is selected
  - $x_i = 0$  means that it is not selected
- Problem constraint
  - -The selected item cannot exceed the capacity of the knapsack



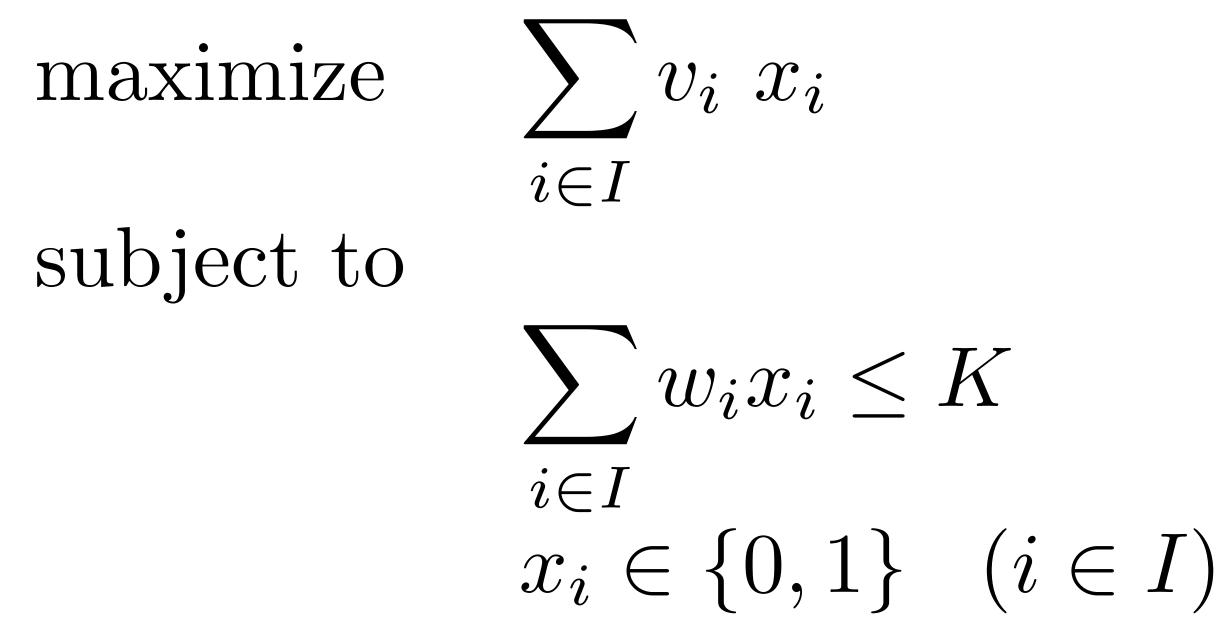
### Decision variables

- x<sub>i</sub> denotes whether item i is selected in the solution
  - $x_i = 1$  means the item is selected
  - $x_i = 0$  means that it is not selected
- Problem constraint
  - -The selected item cannot exceed the capacity of the knapsack
- Objective function
  - -Captures the total value of the selected items





### Putting it all together



# How many possible configurations? - (0,0,0,...,0), (0,0,0,...,1), ..., (1,1,1,...,1)

- How many possible configurations? - (0,0,0,...,0), (0,0,0,...,1), ..., (1,1,1,...,1)
- Not all of them are feasible
  - They cannot exceed the capacity of the knapsack

- How many possible configurations? -(0,0,0,...,0),(0,0,0,...,1),...,(1,1,1,...,1)
- Not all of them are feasible
  - They cannot exceed the capacity of the knapsack
- How many are they? - 2<sup>III</sup>

7

- How many possible configurations? - (0,0,0,...,0), (0,0,0,...,1), ..., (1,1,1,...,1)
- Not all of them are feasible
  - They cannot exceed the capacity of the knapsack
- How many are they? - 2<sup>III</sup>
- How much time to explore them all?
  - 1 millisecond to test a configuration
  - if  $|\mathbf{I}| = 50$ , it will take
    - 1,285,273,866 centuries

### Widely used optimization technique

- for certain classes of problems
- heavily used in computational biology

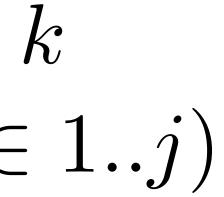
### Widely used optimization technique

- for certain classes of problems
- heavily used in computational biology
- Basic principle
  - divide and conquer
  - bottom up computation

subject to

maximize  $\sum_{i \in 1..j} v_i x_i$ 

 $\sum_{i \in 1...j} w_i x_i \le k$  $x_i \in \{0, 1\} \ (i \in 1...j)$ 



maximize subject to

 $\sum_{i\in 1...j} v_i x_i$ 

 $\sum_{i\in 1...j} w_i x_i \le k$  $x_i \in \{0, 1\} \ (i \in 1..j)$ 

- Basic conventions and notations
  - assume that  $I = \{1, 2, ..., n\}$
  - O(k,j) denotes the optimal solution to the knapsack problem with capacity k and items [1..j]

maximize subject to

 $\sum_{i\in 1...j} v_i x_i$ 

 $\sum_{i\in 1..j} w_i x_i \le k$  $x_i \in \{0, 1\} \ (i \in 1..j)$ 

### Basic conventions and notations

- assume that  $I = \{1, 2, ..., n\}$
- O(k,j) denotes the optimal solution to the knapsack problem with capacity k and items [1..j]

We are interested in finding out the best value O(K,n)

### Assume that we know how to solve

• O(k,j-1) for all k in 0..K

- Assume that we know how to solve • O(k,j-1) for all k in 0..K
- ► We want to solve O(k,j)

• We are just considering one more item, i.e., item j.

- Assume that we know how to solve • O(k,j-1) for all k in 0..K
- ► We want to solve O(k,j)
  - We are just considering one more item, i.e., item j.
- If  $w_i \le k$ , there are two cases
  - Either we do not select item j, then the best solution we can obtain is O(k,j-1)
  - Or we select item j and the best solution is  $v_i + O(k-w_i,j-1)$

- Assume that we know how to solve • O(k,j-1) for all k in 0..K
- ► We want to solve O(k,j)
  - We are just considering one more item, i.e., item j.
- If  $w_i \le k$ , there are two cases
  - Either we do not select item j, then the best solution we can obtain is O(k,j-1)
  - Or we select item j and the best solution is  $v_i + O(k-w_i,j-1)$
- In summary
  - $O(k,j) = max(O(k,j-1), v_i + O(k-w_i,j-1))$  if  $w_i \le k$
  - O(k,j) = O(k,j-1) otherwise

- Assume that we know how to solve • O(k,j-1) for all k in 0..K
- ► We want to solve O(k,j)
  - We are just considering one more item, i.e., item j.
- If  $w_i \le k$ , there are two cases
  - Either we do not select item j, then the best solution we can obtain is O(k,j-1)
  - Or we select item j and the best solution is  $v_i + O(k-w_i,j-1)$
- In summary
  - $O(k,j) = max(O(k,j-1), v_j + O(k-w_j,j-1))$  if  $w_j \le k$
  - O(k,j) = O(k,j-1) otherwise
- ► Of course
  - O(k,0) = 0 for all k

### Recurrence Relations

### We can write a simple program

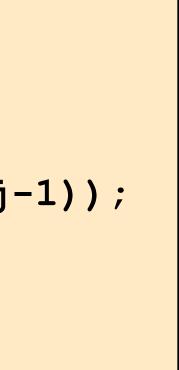
### How efficient is this approach?

Thursday, 13 June 13

### Recurrence Relations

### We can write a simple program

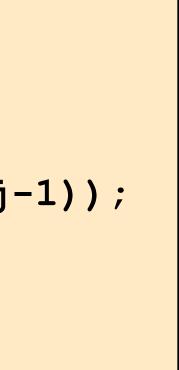
```
int O(int k,int j) {
if (j == 0)
 return 0;
else if (w_j \le k)
  return max(O(k, j-1), v_j + O(k-w_j, j-1));
else
  return O(k,j-1)
```



### Recurrence Relations

### We can write a simple program

```
int O(int k,int j) {
if (j == 0)
 return 0;
else if (w_j \le k)
  return max(O(k, j-1), v_j + O(k-w_j, j-1));
else
  return O(k,j-1)
```



# Recurrence Relations – Fibonacci Numbers

We can write a simple program for finding fibonacci numbers

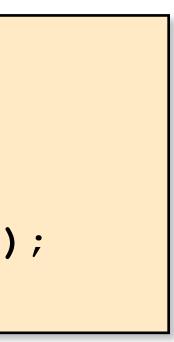
- -we are solving many times the same subproblem
  - fib(n-1) requires fib(n-2) which we have already solved
  - fib(n-3) requires fib(n-4) which we have already solved

# Recurrence Relations – Fibonacci Numbers

### We can write a simple program for finding fibonacci numbers

```
int fib(int n) {
if (n == 0 || n == 1)
   return 1;
else
    return fib(n-2) + fib(n-1);
```

- -we are solving many times the same subproblem
  - fib(n-1) requires fib(n-2) which we have already solved
  - fib(n-3) requires fib(n-4) which we have already solved



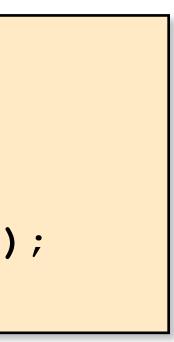
### Recurrence Relations – Fibonacci Numbers

#### We can write a simple program for finding fibonacci numbers

```
int fib(int n) {
if (n == 0 || n == 1)
   return 1;
else
    return fib(n-2) + fib(n-1);
```

#### How efficient is this approach?

- -we are solving many times the same subproblem
  - fib(n-1) requires fib(n-2) which we have already solved
  - fib(n-3) requires fib(n-4) which we have already solved



#### Compute the recursive equations bottom up

- start with zero items
- continue with one item
- then two items
- ...
- then all items

#### How to find which items to select?

Capacity	0
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

#### How to find which items to select?

Capacity	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

#### How to find which items to select?

Capacity	0	1
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
		$v_1 = 5$

$$w_1 = 4$$

#### How to find which items to select?

Capacity	0	1
0	0	0
1	0	0
2	0	0
3	0	0
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
		$v_1 = 5$

$$w_1 = 4$$

#### How to find which items to select?

Capacity	0	1
0	0	0
1	0	0
2	0	0
3	0	0
4	0	5
5	0	5
6	0	5
7	0	5
8	0	5
9	0	5
		$v_1 = 5  w_1 = 4$

#### How to find which items to select?

Capacity	0	1	2
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	5	
5	0	5	
6	0	5	
7	0	5	
8	0	5	
9	0	5	
		$v_1 = 5  w_1 = 4$	$v_2 = 6 \\ w_2 = 5$

#### How to find which items to select?

Capacity	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	5	
5	0	5	
6	0	5	
7	0	5	
8	0	5	
9	0	5	
		$v_1 = 5  w_1 = 4$	$v_2 = 6 \\ w_2 = 5$

#### How to find which items to select?

Capacity	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	5	5
5	0	5	
6	0	5	
7	0	5	
8	0	5	
9	0	5	
$\begin{array}{cccc} v_1 \!=\! 5 & v_2 \!=\! 6 \\ w_1 \!=\! 4 & w_2 \!=\! 5 \end{array}$			

#### How to find which items to select?

Capacity	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	5	5
5	0	5	6
6	0	5	6
7	0	5	6
8	0	5	6
9	0	5	
$v_1 = 5  v_2 = 6 \\ w_1 = 4  w_2 = 5$			

#### How to find which items to select?

Capacity	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	5	5
5	0	5	6
6	0	5	6
7	0	5	6
8	0	5	6
9	0	5	11
$v_1 = 5  v_2 = 6 \\ w_1 = 4  w_2 = 5$			

#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	5	5	
5	0	5	6	
6	0	5	6	
7	0	5	6	
8	0	5	6	
9	0	5	11	
		$v_1 = 5  w_1 = 4$	$v_2 = 6 \\ w_2 = 5$	$v_3 = 3$ $w_3 = 2$





#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	
3	0	0	0	
4	0	5	5	
5	0	5	6	
6	0	5	6	
7	0	5	6	
8	0	5	6	
9	0	5	11	
		$v_1 = 5 \\ w_1 = 4$	$v_2 = 6 \\ w_2 = 5$	$v_3 = 3$ $w_3 = 2$





#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	
5	0	5	6	
6	0	5	6	
7	0	5	6	
8	0	5	6	
9	0	5	11	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				





#### How to find which items to select?

Capacity	0	1	2	3	
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	3	
3	0	0	0	3	
4	0	5	5 🔶		
5	0	5	6		
6	0	5	6		
7	0	5	6		
8	0	5	6		
9	0	5	11		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					





#### How to find which items to select?

Capacity	0	1	2	3	
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	3	
3	0	0	0	3	
4	0	5	5	5	
5	0	5	6	6	
6	0	5	6		
7	0	5	6		
8	0	5	6		
9	0	5	11		
$v_1 = 5  v_2 = 6  v_3 = 3$ $w_1 = 4  w_2 = 5  w_3 = 2$					





#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	
7	0	5	6	
8	0	5	6	
9	0	5	11	
		$v_1 = 5  w_1 = 4$	$v_2 = 6 \\ w_2 = 5$	$v_3 = 3$ $w_3 = 2$





#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	11
		$v_1 = 5  w_1 = 4$	$v_2 = 6 \\ w_2 = 5$	$v_3 = 3$ $w_3 = 2$





#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	11
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				



#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	11
$v_1 = $ <b>Trace back</b>				



#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11←	— 11
$v_1 = $ <b>Trace back</b>				



#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	— 11
$v_1 = $ <b>Trace back</b>				



#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	— 11
$v_1 = $ <b>Trace back</b>				



#### How to find which items to select?

Capacity	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	3
4	0	5	5	5
5	0	5	6	6
6	0	5	6	8
7	0	5	6	9
8	0	5	6	9
9	0	5	11	— 11
Take items 1 and 2 Trace back				



## maximize

maximize

Capacity	0	1	2	3	
0					
1					
2					
3					
4					
5					
6					
7					



maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16	16	16	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16	16	16	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16	16	16	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16	16 🗲	<u>     16     </u>	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16 🔶	<b>—</b> 16 <b>←</b>	<u>    16    </u>	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16 🔶	<b>—</b> 16 <b>←</b>	<u>    16    </u>	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

maximize  $16x_1 + 19x_2 + 23x_3 + 28x_4$ subject to  $2x_1 + 3x_2 + 4x_3 + 5x_4 \le 7$  $x_i \in \{0, 1\} \ (i \in 1..4)$ 

 $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$ 

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16 🔶	<b>—</b> 16 <b>←</b>	<b>— 16</b>	16
3	0	16	19	19	19
4	0	16	19	23	23
5	0	16	35	35	35
6	0	16	35	39	39
7	0	16	35	42	44

#### What is the complexity of this algorithm?

- -time to fill the table
- -i.e., O(K n)

- What is the complexity of this algorithm? -time to fill the table
  - -i.e., O(K n)
- Is this polynomial?

- What is the complexity of this algorithm? -time to fill the table -i.e., O(K n)
- Is this polynomial?
  - -How many bits does K need to be represented on a computer?
    - log(K) bits

- What is the complexity of this algorithm? -time to fill the table -i.e., O(K n)
- Is this polynomial?
  - -How many bits does K need to be represented on a computer?
    - log(K) bits
  - Hence the algorithm is in fact exponential in terms of the input size
    - pseudo-polynomial algorithm
    - "efficient" when K is small

### Until Next Time

# Discrete Optimization

The Knapsack Problem: Part II