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 $i \in$ `I` characterized by
  - its weight w_i
  - its value v_i

The (1-Dimensional) Knapsack Problem

- Given a set of items **I**, each item i ∈ **I** characterized by
  - its weight w_i
  - its value v_i

and
  - a capacity K for a knapsack

The (1-Dimensional) Knapsack Problem

- Given a set of items `I`, each item $i \in$ `I` characterized by
  - its weight w_i
  - its value v_i

 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

Optimization Models

Optimization Models

- How to model an optimization problem

Optimization Models

- How to model an optimization problem
  - choose some decision variables
    - they typically encode the result we are interested into

Optimization Models

- 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

Optimization Models

- 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

Optimization Models

- 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

Optimization Models

- 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”

Optimization Models

- 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

A Knapsack Model

- Decision variables
  - x_i 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

A Knapsack Model

- Decision variables
  - x_i 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 $$\sum_{i \in I} w_i x_i \leq K$$

A Knapsack Model

- Decision variables
  - x_i 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 $$\sum_{i \in I} w_i x_i \leq K$$
- Objective function
  - Captures the total value of the selected items $$\sum_{i \in I} v_i x_i$$

A Knapsack Model

- Putting it all together

$$
\begin{aligned}
&\text{maximize} && \sum_{i \in I} v_i \; x_i \\
&\text{subject to} && \\
& && \sum_{i \in I} w_i x_i \leq K \\
& && x_i \in \{0, 1\} \quad (i \in I)
\end{aligned}
$$

Exponential Growth

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

Exponential Growth

- 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

Exponential Growth

- 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?
  - $2^{|I|}$

Exponential Growth

- 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?
  - $2^{|I|}$
- How much time to explore them all?
  - 1 millisecond to test a configuration
  - if $|I|$ = 50, it will take 1,285,273,866 centuries

Dynamic Programming

- Widely used optimization technique
  - for certain classes of problems
  - heavily used in computational biology

Dynamic Programming

- Widely used optimization technique
  - for certain classes of problems
  - heavily used in computational biology

- Basic principle
  - divide and conquer
  - bottom up computation

Dynamic Programming

$$
\begin{array}{ll}
\text{maximize} & \sum_{i \in 1..j} v_i \; x_i \\
\text{subject to} & \\
& \sum_{i \in 1..j} w_i x_i \leq k \\
& x_i \in \{0,1\} \;\; (i \in 1..j)
\end{array}
$$

Dynamic Programming

$$\begin{array}{ll} \text{maximize} & \sum_{i \in 1..j} v_i \; x_i \\ \text{subject to} & \\ & \sum_{i \in 1..j} w_i x_i \leq k \\ & x_i \in \{0, 1\} \;\; (i \in 1..j) \end{array}$$

- 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]

Dynamic Programming

$$
\begin{array}{ll}
\text{maximize} & \sum_{i \in 1..j} v_i \; x_i \\
\text{subject to} & \\
& \sum_{i \in 1..j} w_i x_i \leq k \\
& x_i \in \{0, 1\} \;\; (i \in 1..j)
\end{array}
$$

- 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)

Recurrence Relations (Bellman Equations)

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

Recurrence Relations (Bellman Equations)

- 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.

Recurrence Relations (Bellman Equations)

- 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 $w_j \leq k$, 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 $v_j + O(k-w_j,j-1)$

Recurrence Relations (Bellman Equations)

- 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 $w_j \leq k$, 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 $v_j + O(k-w_j,j-1)$
- In summary
  - $O(k,j) = \max(O(k,j-1), v_j + O(k-w_j,j-1))$ if $w_j \leq k$
  - $O(k,j) = O(k,j-1)$ otherwise

Recurrence Relations (Bellman Equations)

- 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 $w_j \leq k$, 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 $v_j + O(k-w_j,j-1)$
- In summary
  - $O(k,j) = \max(O(k,j-1), v_j + O(k-w_j,j-1))$ if $w_j \leq k$
  - $O(k,j) = O(k,j-1)$ otherwise
- Of course
  - $O(k,0) = 0$ for all k

Recurrence Relations

- We can write a simple program

- How efficient is this approach?

Recurrence Relations

- We can write a simple program

```
int O(int k,int j) {
   if (j == 0)
     return 0;
   else if (wj <= k)
     return max(O(k,j-1),vj + O(k-wj,j-1));
   else
     return O(k,j-1)
}
```

- How efficient is this approach?

Recurrence Relations

- We can write a simple program

```
int O(int k,int j) {
   if (j == 0)
     return 0;
   else if (wj <= k)
     return max(O(k,j-1),vj + O(k-wj,j-1));
   else
     return O(k,j-1)
}
```

- How efficient is this approach?

Recurrence Relations - Fibonacci Numbers

- We can write a simple program for finding fibonacci numbers

- 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

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

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

Dynamic Programming

- Compute the recursive equations bottom up
  - start with zero items
  - continue with one item
  - then two items
  - ...
  - then all items

Dynamic Programming - Example

- How to find which items to select?

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

Dynamic Programming - Example

- 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

Dynamic Programming - Example

- 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	

$v_1 = 5$
$w_1 = 4$

Dynamic Programming - Example

- 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	

$v_1 = 5$
$w_1 = 4$

Dynamic Programming - Example

- 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

$v_1 = 5$
$w_1 = 4$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$

Dynamic Programming - Example

- 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
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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	
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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
		$v_1 = 5$ $w_1 = 4$	$v_2 = 6$ $w_2 = 5$	$v_3 = 3$ $w_3 = 2$

Dynamic Programming - Example

- 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

$v_1 =$

$w_1 =$

Trace back

Dynamic Programming - Example

- 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

$v_1 =$
$w_1 =$

Trace back

Dynamic Programming - Example

- 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

$v_1 =$
$w_1 =$

Trace back

Dynamic Programming - Example

- 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

$v_1 =$

$w_1 =$

Trace back

Dynamic Programming - Example

- 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

Dynamic Programming - Example

$$
\begin{array}{ll}
\text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\
\text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\
 & x_i \in \{0, 1\} \quad (i \in 1..4)
\end{array}
$$

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0, 1\} \quad (i \in 1..4) \end{array}$$

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

Dynamic Programming - Example

$$\begin{aligned} \text{maximize} \quad & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} \quad & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0, 1\} \quad (i \in 1..4) \end{aligned}$$

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

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0, 1\} \quad (i \in 1..4) \end{array}$$

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

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0,1\} \quad (i \in 1..4) \end{array}$$

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

Dynamic Programming - Example

$$
\begin{array}{ll}
\text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\
\text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\
& x_i \in \{0, 1\} \quad (i \in 1..4)
\end{array}
$$

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

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0,1\} \quad (i \in 1..4) \end{array}$$

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

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0,1\} \quad (i \in 1..4) \end{array}$$

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

Dynamic Programming - Example

$$\begin{array}{ll} \text{maximize} & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\ \text{subject to} & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\ & x_i \in \{0,1\} \quad (i \in 1..4) \end{array}$$

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

Capacity	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	16	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

Dynamic Programming

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

Dynamic Programming

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

- Is this polynomial?

Dynamic Programming

- 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

Dynamic Programming

- 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