Introduction to Algorithms and Data Structures

(Fully) Polynomial-time Approximation Schemes
Methods for approximation algorithms

- Greedy algorithms
- Pricing method (also known as the Primal-Dual method)
- Linear Programming and Rounding
- Dynamic Programming on rounded inputs
Methods for approximation algorithms

- Greedy algorithms
- Pricing method (also known as the Primal-Dual method)
- Linear Programming and Rounding
- Dynamic Programming on rounded inputs
The 0/1-knapsack problem

- We are given a set of $n$ items $\{1, 2, \ldots, n\}$.

- Each item $i$ has a non-negative weight $w_i$ and a non-negative value $v_i$.

- We are given a bound $W$.

- Goal: Select a subset $S$ of the items such that

$$\sum_{i \in S} w_i \leq W$$

and

$$\sum_{i \in S} v_i$$

is maximised.
3 minute exercise

Design a dynamic programming algorithm for 0/1 knapsack.

Algorithm $\text{SubsetSum}(n,W)$

Array $M = [0 \ldots n, 0 \ldots W]$
Initialise $M[0, w] = 0$, for each $w = 0, 1, \ldots, W$

For $i = 1, 2, \ldots, n$
  For $w = 0, \ldots, W$
    If ($w_i > w$)
      $M[i, w] = M[i-1, w]$
    Else
      $M[i, w] = \text{max} \{ M[i-1, w], w_i + M[i-1, w-w_i] \}$
  EndIf

Return $M[n, W]$
3 minute exercise

Design a dynamic programming algorithm for 0/1 knapsack.

Algorithm \textbf{SubsetSum}(n,W)

Array \(M=\left[0 \ldots n, 0 \ldots W\right]\)

Initialise \(M[0, w] = 0\), for each \(w = 0, 1, \ldots, W\)

For \(i = 1, 2, \ldots, n\)

For \(w = 0, \ldots, W\)

If \((w_i > w)\)

\(M[i, w] = M[i-1, w]\)

Else

\(M[i, w] = \max\{M[i-1, w], v_i + M[i-1, w-w_i]\}\)

EndIf

Return \(M[n, W]\)
The dynamic programming algorithm for 0/1 knapsack solves knapsack optimally in time polynomial in $n$ and $W$.

Algorithm \texttt{Knapsack}(n,W, V)

Array $M=[0 \ldots n, 0 \ldots W]$
Initialise $M[0, w] = 0$, for each $w = 0, 1, \ldots, W$

For $i = 1, 2, \ldots, n$
  For $w = 0, \ldots, W$
    If ($w_i > w$)
      $M[i, w] = M[i-1, w]$
    Else
      $M[i, w] = \max\{M[i-1, w], v_i + M[i-1, w-w_i]\}$
  EndIf

Return $M[n, W]$
Another pseudopolynomial time algorithm for 0/1-Knapsack

Algorithm $\text{Knapsack}(n, W, V)$

Array $M=[0 \ldots n, 0 \ldots V]$
Initialise $M[i, 0] = 0$, for $i = 0, 1, \ldots, n$

For $i = 1, 2, \ldots, n$
    For $v = 1, \ldots, \sum_{j=1}^{i} v_j$
        If ($v > \sum_{j=1}^{i-1} v_j$)
            $M[i, v] = w_i + M[i-1, v]$
        Else
            $M[i, v] = \max\{M[i-1, v], w_i + M[i-1, \max(0, v-v_i)]\}$
        EndIf

Return the maximum value $v$ such that $M[n, v] \leq W$. 
Intuition

• We will create subproblems based on the *values*, not the *weights*.

• Each subproblem will be defined by an index $i$ and target value $v$. 
Another pseudopolynomial time algorithm for 0/1-Knapsack

Algorithm Knapsack(n, W, V)

Array M=[0 ... n, 0 ... V]
Initialise M[i, 0] = 0, for i = 0, 1, ..., n

For i = 1, 2, ..., n
   For v = 1, ..., \( \sum_{j=1}^{i-1} v_j \)
      If (v > \( \sum_{j=1}^{i-1} v_j \))
         \[ M[i, v] = w_i + M[i-1, v] \]
      Else
         \[ M[i, v] = \max\{M[i-1, v], w_i + M[i-1, \max(0, v-v_i)]\} \]
   EndIf

Return the maximum value v such that M[n, v] ≤ W.
Intuition

- We will create subproblems based on the values, not the weights.

- Each subproblem will be defined by an index $i$ and target value $v$.

- $M(i, v)$ is the smallest knapsack weight $W$ so that it is possible to obtain a solution using a subset of the items $\{1, \ldots, i\}$ with total value at least $v$. 
Intuition

- We will create subproblems based on the values, not the weights.
- Each subproblem will be defined by an index $i$ and target value $v$.
  
  - $M(i, v)$ is the smallest knapsack weight $W$ so that it is possible to obtain a solution using a subset of the items $\{1, \ldots, i\}$ with total value at least $v$.
- How many subproblems can we have?
Intuition

• We will create subproblems based on the values, not the weights.

• Each subproblem will be defined by an index \(i\) and target value \(v\).

  • \(M(i, v)\) is the smallest knapsack weight \(W\) so that it is possible to obtain a solution using a subset of the items \(\{1, \ldots, i\}\) with total value at least \(v\).

• How many subproblems can we have?

  • At most \(O(n^2v^*)\), where \(v^*\) is the maximum value over all the items.
Intuition

- We will create subproblems based on the values, not the weights.

- Each subproblem will be defined by an index \( i \) and target value \( v \).
  - \( M(i, v) \) is the smallest knapsack weight \( W \) so that it is possible to obtain a solution using a subset of the items \( \{1, \ldots, i\} \) with total value at least \( v \).

- How many subproblems can we have?
  - At most \( O(n^2v^*) \), where \( v^* \) is the maximum value over all the items.

What we know for knapsack

• A pseudo-polynomial algorithm for solving the problem exactly (actually, a couple of those).
What we know for knapsack

• A pseudo-polynomial algorithm for solving the problem exactly (actually, a couple of those).

• What about approximation algorithms?
What we know for knapsack

• A pseudo-polynomial algorithm for solving the problem exactly (actually, a couple of those).

• What about approximation algorithms?

• First try Greedy: Greedy can achieve a 2-approximation.
What we know for knapsack

- A **pseudo-polynomial algorithm** for solving the problem exactly (actually, a couple of those).

- What about approximation algorithms?

- **First try Greedy**: Greedy can achieve a 2-approximation.

- Can we do better?
Rounding the values
Rounding the values

• We will use a rounding parameter $b$. 
Rounding the values

• We will use a rounding parameter $b$.

• For each item $i$, let $\hat{v}_i = \lceil v_i/b \rceil$
Rounding the values

• We will use a rounding parameter $b$.

• For each item $i$, let $\hat{v}_i = \lceil v_i / b \rceil$

  • Intuition: We divide all the values by some factor $b$, and then we round up the result to get integer numbers.
Rounding the values

• We will use a rounding parameter $b$.

• For each item $i$, let $\hat{v}_i = \lceil v_i/b \rceil$

  • Intuition: We divide all the values by some factor $b$, and then we round up the result to get integer numbers.

  • Denote $\tilde{v}_i = \hat{v}_i \cdot b$
Rounding the values

• We will use a rounding parameter $b$.

• For each item $i$, let $\hat{v}_i = \lceil v_i/b \rceil$

  • Intuition: We divide all the values by some factor $b$, and then we round up the result to get integer numbers.

• Denote $\tilde{v}_i = \hat{v}_i \cdot b$

• It holds that for each item $i$, we have $v_i \leq \tilde{v}_i \leq v_i + b$
Why are we doing this?

- Why are we scaling down the values of the knapsack instance?
Why are we doing this?

• Why are we scaling down the values of the knapsack instance?

  • Because we know how to solve the problem in polynomial time when the values are small. How?
Why are we doing this?

- Why are we scaling down the values of the knapsack instance?
  - Because we know how to solve the problem in polynomial time when the values are small. How?
  - We can use our pseudo-polynomial time algorithm.
Why are we doing this?

• Why are we scaling down the values of the knapsack instance?

• Because we know how to solve the problem in polynomial time when the values are small. How?

• We can use our pseudo-polynomial time algorithm.

• But wait, that’s not polynomial, running time was $O(n^2v^*)$. 
Why are we doing this?

• Why are we scaling down the values of the knapsack instance?

  • Because we know how to solve the problem in polynomial time when the values are small. How?

  • We can use our pseudo-polynomial time algorithm.

    • But wait, that’s not polynomial, running time was $O(n^2v^*)$.

    • It is, when $v^*$ is small (i.e., polynomial in $n$).
How much do we lose?

- We solve the knapsack problem after rounding down the values by a factor $b$. 
How much do we lose?

- We solve the knapsack problem after rounding down the values by a factor $b$.
- Why should this change anything?
How much do we lose?

• We solve the knapsack problem after rounding down the values by a factor $b$.

• Why should this change anything?

  • If we scale down the values, the objective function value (the total value of the knapsack) is scaled down as well.
How much do we lose?

• We solve the knapsack problem after rounding down the values by a factor \( b \).

• Why should this change anything?

  • If we scale down the values, the objective function value (the total value of the knapsack) is scaled down as well.

  • We could substitute \( v_i \) with \( v_i/b \) and get an equivalent problem.
How much do we lose?

• We solve the knapsack problem after rounding down the values by a factor $b$.

• Why should this change anything?

  • If we scale down the values, the objective function value (the total value of the knapsack) is scaled down as well.

  • We could substitute $v_i$ with $v_i/b$ and get an equivalent problem.

  • Not quite, because $\hat{v}_i \neq v_i/b$ but $\hat{v}_i = \lceil v_i/b \rceil = \tilde{v}_i/b$
How much do we lose?

- We solve the knapsack problem after rounding down the values by a factor $b$.

- Why should this change anything?

  - If we scale down the values, the objective function value (the total value of the knapsack) is scaled down as well.

  - We could substitute $v_i$ with $v_i/b$ and get an equivalent problem.

  - Not quite, because $\hat{v}_i \neq v_i/b$ but $\hat{v}_i = \lfloor v_i/b \rfloor = \tilde{v}_i/b$

  this is not necessarily an integer
How much do we lose?

- We solve the knapsack problem after rounding down the values by a factor \(b\).

- Why should this change anything?

  - If we scale down the values, the objective function value (the total value of the knapsack) is scaled down as well.

- We could substitute \(v_i\) with \(v_i/b\) and get an equivalent problem.

- Not quite, because \(\hat{v}_i \neq v_i/b\) but \(\hat{v}_i = \lceil v_i/b \rceil = \tilde{v}_i/b\). but this is not necessarily an integer.
How much do we lose?
How much do we lose?

• We need to compare the solutions
How much do we lose?

• We need to compare the solutions
  • when using $v_i$
How much do we lose?

- We need to compare the solutions
  - when using $v_i$
  - when using $\tilde{v}_i$
How much do we lose?

- We need to compare the solutions
  - when using $v_i$
  - when using $\tilde{v}_i$
  - recall: $\tilde{v}_i = \left\lfloor v_i/b \right\rfloor b$
How much do we lose?

• We need to compare the solutions
  • when using $v_i$
  • when using $\tilde{v}_i$
  • recall: $\tilde{v}_i = \left\lfloor \frac{v_i}{b} \right\rfloor b$

• i.e., we need to compute the rounding error.
How much do we lose?

- We need to compare the solutions
  - when using $v_i$
  - when using $\tilde{v}_i$

  - recall: $\tilde{v}_i = \left\lceil \frac{v_i}{b} \right\rceil b$

  - i.e., we need to compute the rounding error.

  - recall: $v_i \leq \tilde{v}_i \leq v_i + b$
How much do we lose?

• We need to compare the solutions
  • when using $v_i$
  • when using $\tilde{v}_i$
  • recall: $\tilde{v}_i = \left\lfloor v_i/b \right\rfloor b$

• i.e., we need to compute the rounding error.
  • recall: $v_i \leq \tilde{v}_i \leq v_i + b$

• the optimal values differ by a factor of $b$. 
The algorithm

Knapsack-Approx(\(\varepsilon\))

Set \(b = (\varepsilon/2n) \max_i v_i\)

Run the DP algorithm for knapsack on values \(\hat{v}_i\)

Return the set \(S\) of items found.
Feasibility

- The set $S$ is a feasible solution to knapsack.
Feasibility

- The set $S$ is a feasible solution to knapsack.
- We didn’t mess up with the weights at all!
Feasibility

- The set $S$ is a feasible solution to knapsack.
  - We didn’t mess up with the weights at all!
  - This is why we could not use the DP algorithm that we knew from previous lectures.
Running Time
Running Time

- The DP algorithm runs in time $O(n^2v^*)$. 
Running Time

• The DP algorithm runs in time $O(n^2v^*)$.

• Recall: $v^* = \max_i v_i$
Running Time

• The DP algorithm runs in time $O(n^2v^*)$.

• Recall: $v^* = \max_i v_i$

• So here, it runs in time polynomial in $n$ and $\max_i \hat{v}_i$
Running Time

• The DP algorithm runs in time $O(n^2 v^*)$.

• Recall: $v^* = \max_i v_i$

• So here, it runs in time polynomial in $n$ and $\max_i \hat{v}_i$

• It holds that: $\arg \max_i v_i = \arg \max_i \hat{v}_i$
Running Time

• The DP algorithm runs in time $O(n^2v^*)$.

• Recall: $v^* = \max_i v_i$

• So here, it runs in time polynomial in $n$ and $\max_i \hat{v}_i$

• It holds that: $\arg\max_i v_i = \arg\max_i \hat{v}_i$

• So we have: $\max_i \hat{v}_i = \lceil \max_i v_i / b \rceil = O(n/\varepsilon)$
Running Time

• The DP algorithm runs in time $O(n^2v^*)$.

• Recall: $v^* = \max_i v_i$

• So here, it runs in time polynomial in $n$ and $\max_i \hat{v}_i$

• It holds that: $\arg\max_i v_i = \arg\max_i \hat{v}_i$

• So we have: $\max_i \hat{v}_i = \lceil \max_i v_i / b \rceil = O(n/\varepsilon)$

$$b = (\varepsilon/2n) \max_i v_i$$
Running Time

- The overall running time is $O(n^3/\varepsilon)$.
- This is polynomial in the input parameters and $1/\varepsilon$. 
Approximation Ratio
Approximation Ratio

• Let $S^*$ be any feasible solution, i.e., any set satisfying

$$\sum_{i \in S^*} w_i \leq W$$
Approximation Ratio

• Let $S^*$ be any feasible solution, i.e., any set satisfying

$$\sum_{i \in S^*} w_i \leq W$$

• We know that $\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i$ (why?)
Approximation Ratio

• Let $S^*$ be any feasible solution, i.e., any set satisfying

$$\sum_{i \in S^*} w_i \leq W$$

• We know that

$$\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i \quad \text{(why?)}$$

• We have the following inequalities:

$$\sum_{i \in S^*} \nu_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (\nu_i + b) \leq nb + \sum_{i \in S} \nu_i$$

$$\nu_i \leq \tilde{v}_i \leq \nu_i + b$$
Approximation Ratio
Approximation Ratio

• Recall: \( b = (\epsilon/2n) \max_i v_i \)
Approximation Ratio

- Recall: \( b = (\varepsilon/2n) \max_{i} v_{i} \)

- Let \( v_{j} \) be the largest value. We have that \( v_{j} = 2nb/\varepsilon \)
Approximation Ratio

- Recall: \[ b = \left( \frac{\varepsilon}{2n} \right) \max_i v_i \]

- Let \( v_j \) be the largest value. We have that \( v_j = \frac{2nb}{\varepsilon} \)

- We also have that \( v_j = \tilde{v}_j \)
Approximation Ratio

• Recall: \[ b = \left( \frac{\varepsilon}{2n} \right) \max_i v_i \]

• Let \( v_j \) be the largest value. We have that \( v_j = \frac{2nb}{\varepsilon} \)

• We also have that \( v_j = \tilde{v}_j \)

\[ \tilde{v}_j = \lceil v_j/b \rceil b = \lceil 2n/\varepsilon \rceil b \]

Assume for simplicity that \( 1/\varepsilon \) is an integer.
Approximation Ratio

- Recall: \( b = (\epsilon/2n) \max_i v_i \)

- Let \( v_j \) be the largest value. We have that \( v_j = 2nb/\epsilon \)

- We also have that \( v_j = \tilde{v}_j \)

- Assumption: Each item fits in the knapsack

\[ \tilde{v}_j = \lceil v_j/b \rceil b = \lceil 2n/\epsilon \rceil b \]

Assume for simplicity that \( 1/\epsilon \) is an integer.
Approximation Ratio

• Recall: \[ b = (\varepsilon/2n) \max_i v_i \]

• Let \( v_j \) be the largest value. We have that \( v_j = 2nb/\varepsilon \)

• We also have that \( v_j = \tilde{v}_j \)

• Assumption: Each item fits in the knapsack

• This implies \( \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = v_j = 2nb/\varepsilon \)

\[ \tilde{v}_j = \lceil v_j/b \rceil b = \lceil 2n/\varepsilon \rceil b \]

Assume for simplicity that \( 1/\varepsilon \) is an integer.
Approximation Ratio

• Recall: 

\[ b = \left(\frac{\varepsilon}{2n}\right) \max_i v_i \]

• Let \( v_j \) be the largest value. We have that \( v_j = \frac{2nb}{\varepsilon} \)

• We also have that \( v_j = \tilde{v}_j \)

Assumption: Each item fits in the knapsack

• This implies 

\[ \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = v_j = \frac{2nb}{\varepsilon} \]

• Finally, from the inequalities of the previous slide, we have

\[ \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq \left(2\varepsilon^{-1} - 1\right)nb \]
Approximation Ratio

• Recall: 

\[ b = \left(\frac{\varepsilon}{2n}\right) \max v_i \]

• Let \( v_j \) be the largest value. We have that 

\[ v_j = \frac{2nb}{\varepsilon} \]

• We also have that \( v_j = \tilde{v}_j \)

• Assumption: Each item fits in the knapsack

• This implies 

\[ \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = v_j = \frac{2nb}{\varepsilon} \]

• Finally, from the inequalities of the previous slide, we have 

\[ \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\varepsilon^{-1} - 1)nb \]
### Approximation Ratio

- **Recall:**
  \[ b = \left(\frac{\varepsilon}{2n}\right) \max_i v_i \]

- Let \( v_j \) be the largest value. We have that \( v_j = 2nb/\varepsilon \)

- We also have that \( v_j = \tilde{v}_j \)

- Assumption: Each item fits in the knapsack
  
  - This implies \( \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = v_j = 2nb/\varepsilon \)
  
  - Finally, from the inequalities of the previous slide, we have
    \[
    \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\varepsilon^{-1} - 1)nb
    \]

\[ \tilde{v}_j = \left\lceil \frac{v_j}{b} \right\rceil b = \left\lceil \frac{2n}{\varepsilon} \right\rceil b \]

Assume for simplicity that \( 1/\varepsilon \) is an integer.
Approximation Ratio

- Recall: \( b = (\epsilon/2n) \max_i v_i \)

- Let \( v_j \) be the largest value. We have that \( v_j = 2nb/\epsilon \)

- We also have that \( v_j = \tilde{v}_j \)

- Assumption: Each item fits in the knapsack

  - This implies \( \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = v_j = 2nb/\epsilon \)

- Finally, from the inequalities of the previous slide, we have
  \[
  \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\epsilon^{-1} - 1)nb
  \]
Approximation Ratio

- Finally, from the inequalities of the previous slide, we have

\[ \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\varepsilon^{-1} - 1)nb \]
Approximation Ratio

- Finally, from the inequalities of the previous slide, we have
  \[ \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\varepsilon^{-1} - 1)nb \]
- From this, for \( \varepsilon \leq 1 \) we have that
  \[ nb \leq \varepsilon \sum_{i \in S} v_i \]
Approximation Ratio

• Finally, from the inequalities of the previous slide, we have

$$\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\varepsilon^{-1} - 1)nb$$

• From this, for $\varepsilon \leq 1$ we have that $nb \leq \varepsilon \sum_{i \in S} v_i$

• Back to the inequalities:

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) \leq nb + \sum_{i \in S} v_i \leq (1 + \varepsilon) \sum_{i \in S} v_i$$
Approximation Ratio

• Finally, from the inequalities of the previous slide, we have

\[ \sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\epsilon^{-1} - 1)nb \]

• From this, for \( \epsilon \leq 1 \) we have that \( nb \leq \epsilon \sum_{i \in S} v_i \)

• Back to the inequalities:

\[ \sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) \leq nb + \sum_{i \in S} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i \]
Finally, from the inequalities of the previous slide, we have

$$\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \Rightarrow \sum_{i \in S} v_i \geq (2\epsilon^{-1} - 1)nb$$

From this, for $\epsilon \leq 1$ we have that $nb \leq \epsilon \sum_{i \in S} v_i$

Back to the inequalities:

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) \leq nb + \sum_{i \in S} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$
PTAS vs FPTAS

• **PTAS (Polynomial Time Approximation Scheme):** An approximation algorithm which, given an $\varepsilon$, runs in time polynomial in the input parameters and has approximation ratio $1+\varepsilon$.

• **FPTAS (Fully Polynomial Time Approximation Scheme):** An approximation algorithm which, given an $\varepsilon$, runs in time polynomial in the input parameters and $1/\varepsilon$ and has approximation ratio $1+\varepsilon$. 
PTAS vs FPTAS

• **PTAS (Polynomial Time Approximation Scheme):** An approximation algorithm which, given an $\varepsilon$, runs in time polynomial in the input parameters and has approximation ratio $1+\varepsilon$.

• **FPTAS (Fully Polynomial Time Approximation Scheme):** An approximation algorithm which, given an $\varepsilon$, runs in time polynomial in the input parameters and $1/\varepsilon$ and has approximation ratio $1+\varepsilon$.

• What is the algorithm that we designed for knapsack? A PTAS or an FPTAS?
A PTAS (sketch) for knapsack
A PTAS (sketch) for knapsack

• Consider all possible subsets of items with size at most $k$. 
A PTAS (sketch) for knapsack

- Consider all possible subsets of items with size at most $k$.
  
  - There are $O(kn^k)$ of those.
A PTAS (sketch) for knapsack

• Consider all possible subsets of items with size at most $k$.

  • There are $O(kn^k)$ of those.

  • For each one of those subsets, put those items in the knapsack, and use a greedy algorithm to fill up the rest of the knapsack.
A PTAS (sketch) for knapsack

• Consider all possible subsets of items with size at most $k$.
  
  • There are $O(kn^k)$ of those.
  
  • For each one of those subsets, put those items in the knapsack, and use a greedy algorithm to fill up the rest of the knapsack.
  
  • One can prove that this solution is a $1+1/k$ approximation in time $O(kn^{k+1})$. 
A PTAS (sketch) for knapsack

- Consider all possible subsets of items with size at most $k$.
  
  - There are $O(kn^k)$ of those.
  
  - For each one of those subsets, put those items in the knapsack, and use a greedy algorithm to fill up the rest of the knapsack.

- One can prove that this solution is a $1+1/k$ approximation in time $O(kn^{k+1})$.

- We can pick $\varepsilon=1/k$, and we have a $1+\varepsilon$ approximation in time $O((1/\varepsilon)n^{1/\varepsilon})$. 
A PTAS (sketch) for knapsack

- Consider all possible subsets of items with size at most $k$.
  - There are $O(kn^k)$ of those.
  - For each one of those subsets, put those items in the knapsack, and use a greedy algorithm to fill up the rest of the knapsack.
  - One can prove that this solution is a $1+1/k$ approximation in time $O(kn^{k+1})$.
  - We can pick $\varepsilon=1/k$, and we have a $1+\varepsilon$ approximation in time $O((1/\varepsilon)n^{1/\varepsilon})$.
  - This is polynomial in $n$ but not in $1/\varepsilon$. 
Inapproximability

• **Definition:** A problem $P$ is *strongly* NP-hard, when there is a polynomial time reduction from a *strongly* NP-hard problem to it.

• For a *strongly* NP-hard problem $P$,
  
  - There is **no** Fully Polynomial Time Approximation Scheme (FPTAS).
  
  - There is **no** pseudo-polynomial time algorithm that solves it exactly.
Approximation algorithms:
A big chapter
Approximation algorithms: A big chapter

- Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, …)
Approximation algorithms: A big chapter

- Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, …)

- Limitations of algorithms (tight instances).
Approximation algorithms: A big chapter

- Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, ...)

- Limitations of algorithms (tight instances).

- Limitations of techniques (e.g., integrality gap).
Approximation algorithms: A big chapter

- Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, …)

- Limitations of algorithms (tight instances).

- Limitations of techniques (e.g., integrality gap).

- Inapproximability
Approximation algorithms: A big chapter

• Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, …)

• Limitations of algorithms (tight instances).

• Limitations of techniques (e.g., integrality gap).

• Inapproximability

  • How do we prove this?
Approximation algorithms: A big chapter

- Different techniques (greedy, pricing method aka primal-dual, LP-relaxation and rounding, DP on rounded inputs, brute-force and greedy, dual fitting, dual LP-relaxation and rounding, ...)

- Limitations of algorithms (tight instances).

- Limitations of techniques (e.g., integrality gap).

- Inapproximability
  
  - How do we prove this?

  - Sometimes easy, sometimes hard, mostly hard!
Reading

- Kleinberg and Tardos 11.8.
- Williamson and Shmoys 3.1 (slightly different exposition).
That’s all from me!
That’s all from me!

• Thank you everyone for attending the lectures and being engaged with the course.
That’s all from me!

- Thank you everyone for attending the lectures and being engaged with the course.

- There’s more to come (John’s lectures on the fascinating topic of undecidability, assignment, quiz, tutorials, labs, possibly a revision class, and of course the exam).
That’s all from me!

• Thank you everyone for attending the lectures and being engaged with the course.

• There’s more to come (John’s lectures on the fascinating topic of undecidability, assignment, quiz, tutorials, labs, possibly a revision class, and of course the exam).

• If you are interested in learning more about algorithms, consider taking ADS next year.
That’s all from me!

• Thank you everyone for attending the lectures and being engaged with the course.

• There’s more to come (John’s lectures on the fascinating topic of undecidability, assignment, quiz, tutorials, labs, possibly a revision class, and of course the exam).

• If you are interested in learning more about algorithms, consider taking ADS next year.

• See some of you then!