Greedy Approximations
Approximation Schemes: Knapsack Approximation

Algorithm Design & Analysis

Spring 2018
Announcements

• Colloquium Today: Prof. Martin Farach-Colton (Rutgers): Spamming PageRank
• Problem Set 9: For practice only; no need to submit it
• Change of Topic for Next Week: Lower Bounds
• Course evaluations will be administered next Wednesday!
Announcements

• Colloquium Today: Prof. Martin Farach-Colton (Rutgers): Spamming PageRank
Announcements

- Colloquium Today: Prof. Martin Farach-Colton (Rutgers): Spamming PageRank
- Problem Set 9: For practice only; no need to submit it
- Change of Topic for Next Week: Lower Bounds
- Course evaluations will be administered next Wednesday!
Announcements

- Colloquium Today: Prof. Martin Farach-Colton (Rutgers): Spamming PageRank
- Problem Set 9: For practice only; no need to submit it
- Change of Topic for Next Week: Lower Bounds
- Course evaluations will be administered next Wednesday!
Announcements

• Colloquium Today: Prof. Martin Farach-Colton (Rutgers): Spamming PageRank
• Problem Set 9: For practice only; no need to submit it
• Change of Topic for Next Week: Lower Bounds
• Course evaluations will be administered next Wednesday!
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm \textit{UnitGreed}

1. Sort items so that \( \frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n} \)

2. Find largest \( k \) such that

\[
\sum_{i=1}^{k} w_i \leq W
\]

3. If \( \sum_{i=1}^{k} v_i > v_{k+1} \) take items 1, \ldots, \( k \), otherwise take item \( k + 1 \)
A 2-Approximation for Knapsack

Consider the following greedy knapsack algorithm $UnitGreed$

1. Sort items so that $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$

2. Find largest $k$ such that

$$\sum_{i=1}^{k} w_i \leq W$$

3. If $\sum_{i=1}^{k} v_i > v_{k+1}$ take items 1, \ldots, $k$, otherwise take item $k + 1$

Claim: $UnitGreed$ produces a result within a factor of 2 of the maximum.
**Polynomial Time Approximation Schemes (PTAS)**

**Given:** A maximization problem $P$ and an $c$-approximation algorithm for $P$. For any instance $I$ of $P$ where the approximate solution is sub-optimal, let $opt(I)$ and $approx(I)$ refer to the value of the optimum and approximate solutions to $I$. Then $1 < opt(I)/approx(I) \leq c$, so $c > 1$; that is, $c = 1 + \varepsilon$ for some $\varepsilon > 0$. 

**Question:** Is it possible that, for some NP-hard problems, there are $(1 + \varepsilon)$-approximations for every $\varepsilon > 0$? What, exactly, would that mean?
**Polynomial Time Approximation Schemes (PTAS)**

**Given:** A maximization problem $P$ and an $c$-approximation algorithm for $P$. For any instance $I$ of $P$ where the approximate solution is sub-optimal, let $opt(I)$ and $approx(I)$ refer to the value of the optimum and approximate solutions to $I$. Then $1 < opt(I)/approx(I) \leq c$, so $c > 1$; that is, $c = 1 + \epsilon$ for some $\epsilon > 0$.

**Question:** Is it possible that, for some NP-hard problems, there are $(1+\epsilon)$-approximations for every $\epsilon > 0$?
**Given:** A maximization problem $P$ and an $c$-approximation algorithm for $P$. For any instance $I$ of $P$ where the approximate solution is sub-optimal, let $\text{opt}(I)$ and $\text{approx}(I)$ refer to the value of the optimum and approximate solutions to $I$. Then $1 < \text{opt}(I)/\text{approx}(I) \leq c$, so $c > 1$; that is, $c = 1 + \epsilon$ for some $\epsilon > 0$.

**Question:** Is it possible that, for some NP-hard problems, there are $(1+\epsilon)$-approximations for *every* $\epsilon > 0$?

**Question:** What, exactly, would that mean?
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small.
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$

- Denote this weight by $\overline{opt}(i, V)$, where $V = 0, \ldots, \sum_{j=1}^{i} v_j$
- Note: $\overline{opt}(i, V)$ increases as $i$ decreases and as $V$ increases
- Note: If $v^* = \max_i v_i$, then only consider $V \leq nv^*$
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$

- Denote this weight by $\overline{\text{opt}}(i, V)$, where $V = 0, \ldots, \sum_{j=1}^{i} v_j$
- Note: $\overline{\text{opt}}(i, V)$ increases as $i$ decreases and as $V$ increases
- Note: If $v^* = \max_i v_i$, then only consider $V \leq nv^*$
- Note: $\overline{\text{opt}}(i, V)$ isn't an optimal solution to a sub-problem of Knapsack, but
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$

- Denote this weight by $\overline{opt}(i, V)$, where $V = 0, \ldots, \sum_{j=1}^{i} v_j$
- Note: $\overline{opt}(i, V)$ increases as $i$ decreases and as $V$ increases
- Note: If $v^* = \max_i v_i$, then only consider $V \leq nv^*$
- Note: $\overline{opt}(i, V)$ isn’t an optimal solution to a sub-problem of Knapsack, but
- Optimum knapsack solution is the largest $V$ for which $\overline{opt}(n, V) \leq W$
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$

- Denote this weight by $\overline{opt}(i, V)$, where $V = 0, \ldots, \sum_{j=1}^{i} v_j$
- **Note:** $\overline{opt}(i, V)$ increases as $i$ decreases and as $V$ increases
- **Note:** If $v^* = \max_i v_i$, then only consider $V \leq nv^*$
- **Note:** $\overline{opt}(i, V)$ isn't an optimal solution to a sub-problem of Knapsack, but
- Optimum knapsack solution is the largest $V$ for which $\overline{opt}(n, V) \leq W$
- That is, $\overline{opt}(n, W) = \max_{V} \{\overline{opt}(n, V) \leq W\}$
A Value-Oriented DP Algorithm

DP for Knapsack is Pseudo-Polynomial: Runs in time $O(nW)$

Idea: Develop an algorithm that works well when values are small.

Consider smallest weight using items $\{1, \ldots, i\}$ achieving value at least $V$.

- Denote this weight by $\text{opt}(i, V)$, where $V = 0, \ldots, \sum_{j=1}^{i} v_j$
- **Note:** $\text{opt}(i, V)$ increases as $i$ decreases and as $V$ increases.
- **Note:** If $v^* = \max_i v_i$, then only consider $V \leq nv^*$.
- **Note:** $\text{opt}(i, V)$ isn’t an optimal solution to a sub-problem of Knapsack, but

  - Optimum knapsack solution is the largest $V$ for which $\text{opt}(n, V) \leq W$.
  - That is, $\text{opt}(n, W) = \max V \{ \text{opt}(n, V) \leq W \}$
  - $\text{opt}(-, -)$ has size $n^2 v^*$.
A Recurrence For $\text{opt}(i, V)$

Let $O$ be the optimal solution for the weight minimization problem. Then

- If $n \notin O$, then $\text{opt}(n, V) = \text{opt}(n-1, V)$
- If $n$ is the only item in $O$, then $\text{opt}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then $\text{opt}(n, V) = w_n + \text{opt}(n-1, V-v_n)$

Note: If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds.

- If not, then $\text{opt}(n, V)$ is the smallest of
  - $\text{opt}(n-1, V)$
  - $w_n$
  - $w_n + \text{opt}(n-1, V-v_n)$

So $\text{opt}(-, -)$ can be built in time $O(n^2 v^*)$ (good for small values).
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n$

A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

Note: If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds.

If not, then $\overline{\text{opt}}(n, V)$ is the smallest of $\overline{\text{opt}}(n - 1, V)$, $w_n$, and $\overline{\text{opt}}(n - 1, V - v_n)$.

So $\overline{\text{opt}}(n, V)$ can be built in time $O(n^2 v^*)$, good for small values.
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then
  $$\text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n)$$
- **Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds
- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of

So $\overline{\text{opt}}(\cdot, V)$ can be built in time $O(n^2 v^*)$ (good for small values)
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$

**Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds

- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of
  - $\overline{\text{opt}}(n - 1, V)$
A Recurrence For $\overline{\text{opt}}(i, V)$

Let $O$ be the optimal solution for the weight minimization problem. Then

- If $n \notin O$, then $\overline{\text{opt}}(n, V) = \overline{\text{opt}}(n - 1, V)$
- If $n$ is the only item in $O$, then $\overline{\text{opt}}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then $\overline{\text{opt}}(n, V) = w_n + \overline{\text{opt}}(n - 1, V - v_n)$
- **Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds
- If not, then $\overline{\text{opt}}(n, V)$ is the smallest of
  - $\overline{\text{opt}}(n - 1, V)$
  - $w_n$
A Recurrence For $\text{opt}(i, V)$

Let $O$ be the optimal solution for the weight minimization problem. Then

- If $n \notin O$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $O$, then $\text{opt}(n, V) = w_n$
- If $n \in O$ is not the only item in $O$, then
  \[ \text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n) \]
- **Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds
- If not, then $\text{opt}(n, V)$ is the smallest of
  - $\text{opt}(n - 1, V)$
  - $w_n$
  - $w_n + \text{opt}(n - 1, V - v_n)$
A Recurrence For $\text{opt}(i, V)$

Let $\mathcal{O}$ be the optimal solution for the weight minimization problem. Then

- If $n \notin \mathcal{O}$, then $\text{opt}(n, V) = \text{opt}(n - 1, V)$
- If $n$ is the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n$
- If $n \in \mathcal{O}$ is not the only item in $\mathcal{O}$, then $\text{opt}(n, V) = w_n + \text{opt}(n - 1, V - v_n)$
- **Note:** If $V > \sum_{i=1}^{n-1} v_i$, then it must be that the previous case holds
- If not, then $\text{opt}(n, V)$ is the smallest of
  - $\text{opt}(n - 1, V)$
  - $w_n$
  - $w_n + \text{opt}(n - 1, V - v_n)$
  - So $\text{opt}(\_ , \_)$ can be built in time $O(n^2 v^*)$ (good for small values)
A Polynomial Time Approximation Scheme

Goal: Find a *tunable* approximation algorithm
A Polynomial Time Approximation Scheme

Goal: Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
A Polynomial Time Approximation Scheme

Goal: Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
A Polynomial Time Approximation Scheme

**Goal:** Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a rounding technique

- If values are small, use algorithm just described
- Otherwise, round values up by some $b$: Let $\tilde{v}_i = \left\lceil \frac{v_i}{b} \right\rceil b$
  - Note: $\tilde{v}_i \approx v_i$: $v_i \leq \tilde{v}_i \leq v_i + b$

  Solve rounded problem (actually, solve using $\hat{v}_i = \left\lceil \frac{v_i}{b} \right\rceil$)

  Values are now smaller (Spoiler alert: $b$ depends on $\epsilon$)
A Polynomial Time Approximation Scheme

Goal: Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a *rounding* technique
A Polynomial Time Approximation Scheme

**Goal:** Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a *rounding* technique

- If values are small, use algorithm just described
A Polynomial Time Approximation Scheme

Goal: Find a tunable approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But not polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a rounding technique

- If values are small, use algorithm just described
- Otherwise, round values up by some $b$: Let $\tilde{v}_i = \lceil v_i/b \rceil b$
**A Polynomial Time Approximation Scheme**

**Goal:** Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a *rounding* technique

- If values are small, use algorithm just described
- Otherwise, round values up by some $b$: Let $\tilde{v}_i = \lceil v_i/b \rceil b$
- **Note:** $\tilde{v}_i \approx v_i : v_i \leq \tilde{v}_i \leq v_i + b$
A Polynomial Time Approximation Scheme

**Goal:** Find a *tunable* approximation algorithm
- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a *rounding* technique

- If values are small, use algorithm just described
- Otherwise, round values up by some $b$: Let $\tilde{v}_i = \lceil v_i/b \rceil b$
- **Note:** $\tilde{v}_i \approx v_i : v_i \leq \tilde{v}_i \leq v_i + b$
- Solve rounded problem (actually, solve using $\hat{v}_i = \lceil v_i/b \rceil$)
A Polynomial Time Approximation Scheme

Goal: Find a *tunable* approximation algorithm

- Given $\epsilon > 0$, algorithm will return solution within factor of $1 + \epsilon$ of optimal
- For any fixed $\epsilon$, algorithm is polynomial time.
- But *not* polynomial in $\epsilon$ (!)

We will develop such an algorithm for the knapsack problem via a *rounding* technique

- If values are small, use algorithm just described
- Otherwise, round values up by some $b$: Let $\tilde{v}_i = \lceil v_i / b \rceil b$
- **Note:** $\tilde{v}_i \approx v_i : \quad v_i \leq \tilde{v}_i \leq v_i + b$
- Solve rounded problem (actually, solve using $\hat{v}_i = \lceil v_i / b \rceil$)
- Values are now smaller (Spoiler alert: $b$ depends on $\epsilon$)
Knapsack PTAS

Observe: For weights \(\{w_1, \ldots, w_n\}\), the knapsack problems with values \(\{\hat{v}_1, \ldots, \hat{v}_n\}\) and with values \(\{\tilde{v}_1, \ldots, \tilde{v}_n\}\) have the same sets of optimal solutions.
**Knapsack PTAS**

**Observe:** For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \)

- Let \( b = (\epsilon/2n) \max_i v_i \) (\( b = \max_i v_i / (2n \epsilon - 1) \), \( \epsilon - 1 \in \mathbb{N} \))

- Solve knapsack problem with values \( \hat{v}_i = \lceil v_i / b \rceil \)

- Idea: Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation

- Note: Algorithm runs in time \( O(n^2 \hat{v}^*) \), where \( \hat{v}^* = \max v_i \uparrow \)

- But \( v^* \) came from the maximum \( v_j \), so \( \hat{v}^* = \lceil v_j / b \rceil = 2n \epsilon - 1 \)

- Thus \( O(n^2 \hat{v}^*) = O(n^3 \epsilon - 1) \)

- Where \( \epsilon - 1 \) is a (BIG) constant!
Knapsack PTAS

Observe: For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \)
  - Let \( b = (\epsilon/2n) \max_i v_i \) (\( b = \max_i v_i/(2n\epsilon^{-1}), \epsilon^{-1} \in \mathbb{N} \))
  - Solve knapsack problem with values \( \hat{v}_i = \lceil v_i/b \rceil \)

   Where \( \epsilon^{-1} \) is a (BIG) constant!
Knapsack PTAS

Observe: For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \)
- Let \( b = (\epsilon/2n) \max_i v_i \) \( (b = \max_i v_i/(2n\epsilon^{-1}), \epsilon^{-1} \in \mathbb{N}) \)
- Solve knapsack problem with values \( \hat{v}_i = \lceil v_i/b \rceil \)

- Idea: Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation
Knapsack PTAS

Observe: For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \)
- Let \( b = (\epsilon/2n) \max_i v_i \) (\( b = \max_i v_i/(2n\epsilon^{-1}) \), \( \epsilon^{-1} \in \mathbb{N} \))
- Solve knapsack problem with values \( \hat{v}_i = \lceil v_i/b \rceil \)

- **Idea:** Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation

- **Note:** Algorithm runs in time \( O(n^2 \hat{v}^*) \), where \( v^* = \max_i v_i \)
Knapsack PTAS

Observe: For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \). Let \( b = (\epsilon/2n) \max_i v_i \) (\( b = \max_i v_i/(2n\epsilon^{-1}), \epsilon^{-1} \in \mathbb{N} \)) Solve knapsack problem with values \( \hat{v}_i = \lceil v_i/b \rceil \)

- **Idea:** Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation

- **Note:** Algorithm runs in time \( O(n^2 \hat{v}^*) \), where \( v^* = \max_i v_i \)

- But \( v^* \) came from the maximum \( v_j \), so \( \hat{v}^* = \lceil v_j/b \rceil = 2n\epsilon^{-1} \)
**Knapsack PTAS**

**Observe:** For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- **Delete any items with weight greater than** \( W \)  
  Let 
  \[
  b = (\epsilon/2n) \max_i v_i \quad (b = \max_i v_i/(2n\epsilon^{-1}), \epsilon^{-1} \in \mathbb{N})
  \]
  Solve knapsack problem with values \( \hat{v}_i = \lceil v_i / b \rceil \)

- **Idea:** Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation

- **Note:** Algorithm runs in time \( O(n^2 \hat{v}^*) \), where \( \hat{v}^* = \max_i v_i \)
  - But \( v^* \) came from the maximum \( v_j \), so \( \hat{v}^* = \lceil v_j / b \rceil = 2n\epsilon^{-1} \)
  - Thus \( O(n^2 \hat{v}^*) = O(n^3 \epsilon^{-1}) \)
Knapsack PTAS

Observe: For weights \( \{w_1, \ldots, w_n\} \), the knapsack problems with values \( \{\hat{v}_1, \ldots, \hat{v}_n\} \) and with values \( \{\tilde{v}_1, \ldots, \tilde{v}_n\} \) have the same sets of optimal solutions.

The algorithm is

- Delete any items with weight greater than \( W \)
  
  Let
  
  \[ b = \left( \frac{\epsilon}{2n} \right) \max_i v_i \]  
  \( (b = \max_i v_i / (2n\epsilon^{-1}), \ \epsilon^{-1} \in \mathbb{N}) \)
  
  Solve knapsack problem with values \( \hat{v}_i = \lceil v_i / b \rceil \)

- Idea: Smaller \( \epsilon \) gives smaller \( b \), yielding a better approximation

- Note: Algorithm runs in time \( O(n^2 \hat{v}^*) \), where \( v^* = \max_i v_i \)
  
  But \( v^* \) came from the maximum \( v_j \), so \( \hat{v}^* = \lceil v_j / b \rceil = 2n\epsilon^{-1} \)
  
  Thus \( O(n^2 \hat{v}^*) = O(n^3 \epsilon^{-1}) \)
  
  Where \( \epsilon^{-1} \) is a (BIG) constant!
Complexity of Knapsack PTAS

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$
\sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i
$$
**Complexity of Knapsack PTAS**

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$\sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$

**Proof:**
Complexity of Knapsack PTAS

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$\sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$

**Proof:**

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \tilde{v}_i \leq \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} (v_i + b) = nb + \sum_{i \in S} v_i$$
Complexity of Knapsack PTAS

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$
\sum_{i \in S} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i
$$

**Proof:**

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

Let $v_j = \max_i v_i$, then $v_j = 2nb\epsilon^{-1}$ and $v_j = \tilde{v}_j$. So, if $\epsilon \leq 1$
**Complexity of Knapsack PTAS**

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$\sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$

**Proof:**

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

Let $v_j = \max_i v_i$, then $v_j = 2nb\epsilon^{-1}$ and $v_j = \tilde{v}_j$. So, if $\epsilon \leq 1$

$$\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb \geq \tilde{v}_j - nb \geq 2nb\epsilon^{-1} - nb\epsilon^{-1} \geq nb\epsilon^{-1}$$
Complexity of Knapsack PTAS

**Theorem:** Let $S$ be the solution found by Knapsack PTAS and let $S^*$ be any other solution. Then

$$\sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$

**Proof:**

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

Let $v_j = \max_i v_i$, then $v_j = 2nb\epsilon^{-1}$ and $v_j = \tilde{v}_j$. So, if $\epsilon \leq 1$

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

So,

$$nb \leq \epsilon \sum_{i \in S} v_i, \text{ and so } \sum_{i \in S^*} v_i \leq (1 + \epsilon) \sum_{i \in S} v_i$$
How in the World Would Someone Guess $b = (\epsilon/2n)\max_i v_i$?
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$. 

The author then uses the fact that $\epsilon/2 \leq \epsilon/(1+\epsilon)$, if $\epsilon \leq 1$. 

Since any $b \leq (\epsilon/2n) v_j$ will work, let $b = (\epsilon/2n) v_j = (\epsilon/2n) v^*$.
How in the World Would Someone Guess \( b = (\epsilon/2n) \max_i v_i \)?

Well, we need to pick \( b \) so that \( nb \leq \epsilon \sum_{i \in S} v_i \).

We know that
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$.

We know that

- $\sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i$, and
How in the World Would Someone Guess

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

Well, we need to pick \( b \) so that \( nb \leq \epsilon \sum_{i \in S} v_i \).

We know that

- \( \sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i \), and
- \( \max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i \) (since \( S \) is optimal for rounded)
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$.

We know that

- $\sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i$, and
- $\max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i$ (since $S$ is optimal for rounded)

Combining these gives
How in the World Would Someone Guess 
\[ b = (\epsilon/2n) \max_i v_i? \]

Well, we need to pick \( b \) so that \( nb \leq \epsilon \sum_{i \in S} v_i \).

We know that

- \( \sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i \), and
- \( \max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i \) (since \( S \) is optimal for rounded)

Combining these gives

- \( v_j \leq nb + \sum_{i \in S} v_i \); that is, \( v_j - nb \leq \sum_{i \in S} v_i \)
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$.

We know that

- $\sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i$, and
- $\max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i$ (since $S$ is optimal for rounded)

Combining these gives

- $v_j \leq nb + \sum_{i \in S} v_i$; that is, $v_j - nb \leq \sum_{i \in S} v_i$
- So, $\epsilon (v_j - nb) \leq \epsilon \sum_{i \in S} v_i$
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$.

We know that

- $\sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i$, and
- $\max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i$ (since $S$ is optimal for rounded)

Combining these gives

- $v_j \leq nb + \sum_{i \in S} v_i$; that is, $v_j - nb \leq \sum_{i \in S} v_i$
- So, $\epsilon (v_j - nb) \leq \epsilon \sum_{i \in S} v_i$

Thus, if we can find $b$ so that $nb \leq \epsilon (v_j - nb)$, then we’re done!
How in the World Would Someone Guess \( b = (\epsilon/2n) \max_i v_i \)?

Well, we need to pick \( b \) so that \( nb \leq \epsilon \sum_{i \in S} v_i \).

We know that

- \( \sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i \), and
- \( \max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i \) (since \( S \) is optimal for rounded)

Combining these gives

- \( v_j \leq nb + \sum_{i \in S} v_i \); that is, \( v_j - nb \leq \sum_{i \in S} v_i \)
- So, \( \epsilon(v_j - nb) \leq \epsilon \sum_{i \in S} v_i \)

Thus, if we can find \( b \) so that \( nb \leq \epsilon(v_j - nb) \), then we’re done!

Just solve this inequality for \( b \), giving \( b \leq (\epsilon/(1 + \epsilon) n)v_j \).
How in the World Would Someone Guess 
\[ b = (\epsilon/2n) \max_i v_i? \]

Well, we need to pick \( b \) so that \( nb \leq \epsilon \sum_{i \in S} v_i \).

We know that

- \( \sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i \), and
- \( \max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i \) (since \( S \) is optimal for rounded)

Combining these gives

- \( v_j \leq nb + \sum_{i \in S} v_i \); that is, \( v_j - nb \leq \sum_{i \in S} v_i \)
- So, \( \epsilon(v_j - nb) \leq \epsilon \sum_{i \in S} v_i \)

Thus, if we can find \( b \) so that \( nb \leq \epsilon(v_j - nb) \), then we’re done!

Just solve this inequality for \( b \), giving \( b \leq (\epsilon/(1 + \epsilon)n)v_j \)

The author then uses the fact that \( \epsilon/2 \leq \epsilon/(1 + \epsilon) \), if \( \epsilon \leq 1 \)
How in the World Would Someone Guess $b = (\epsilon/2n) \max_i v_i$?

Well, we need to pick $b$ so that $nb \leq \epsilon \sum_{i \in S} v_i$.

We know that

1. $\sum_{i \in S} \tilde{v}_i \leq nb + \sum_{i \in S} v_i$, and
2. $\max_i v_i = v_j \leq \tilde{v}_j \leq \sum_{i \in S} \tilde{v}_i$ (since $S$ is optimal for rounded)

Combining these gives

1. $v_j \leq nb + \sum_{i \in S} v_i$; that is, $v_j - nb \leq \sum_{i \in S} v_i$
2. So, $\epsilon(v_j - nb) \leq \epsilon \sum_{i \in S} v_i$

Thus, if we can find $b$ so that $nb \leq \epsilon(v_j - nb)$, then we’re done!

Just solve this inequality for $b$, giving $b \leq (\epsilon/(1 + \epsilon)n)v_j$

The author then uses the fact that $\epsilon/2 \leq \epsilon/(1 + \epsilon)$, if $\epsilon \leq 1$

Since any $b \leq (\epsilon/2n)v_j$ will work, let $b = (\epsilon/2n)v_j = (\epsilon/2n)v^*$
Johan Håstad [1999] proved

Theorem

For any $\epsilon > 0$, unless $\text{NP} = \text{P}$, there is no polynomial-time algorithm that approximates MAX_INDEPENDENT_SET within a factor of $n^{1 - \epsilon}$.

In fact, he showed something potentially stronger

Theorem

For any $\epsilon > 0$, unless $\text{NP} = \text{ZPP}$ there is no polynomial-time algorithm that approximates MAX_INDEPENDENT_SET within a factor of $n^{1 - \epsilon}$.

But a graph can have an independent set of size $cn$, so if $\text{opt} > n^{1 - \epsilon}$ then $\text{approx} < cn^{1 - \epsilon} = cn \epsilon \rightarrow c$ as $n \rightarrow \infty$.

So this problem is generally considered to be inapproximable.
Johan Håstad [1999] proved

**Theorem**

For any $\epsilon > 0$, unless $NP = P$, there is no polynomial-time algorithm that approximates MAX\_INDEPENDENT\_SET within a factor of $n^{1/2 - \epsilon}$. 
Johan Håstad [1999] proved

**Theorem**

For any $\epsilon > 0$, unless $\text{NP} = \text{P}$, there is no polynomial-time algorithm that approximates $\text{MAX\_INDEPENDENT\_SET}$ within a factor of $n^{\frac{1}{2} - \epsilon}$.

In fact, he showed something potentially stronger
Johan Håstad [1999] proved

**Theorem**
For any $\epsilon > 0$, unless $NP=\bar{P}$, there is no polynomial-time algorithm that approximates $\text{MAX\_INDEPENDENT\_SET}$ within a factor of $n^{\frac{1}{2} - \epsilon}$.

In fact, he showed something potentially stronger

**Theorem**
For any $\epsilon > 0$, unless $NP=\text{ZPP}$ there is no polynomial-time algorithm that approximates $MAX\_INDEPENDENT\_SET$ within a factor of $n^{1-\epsilon}$.
Johan Håstad [1999] proved

**Theorem**
For any $\epsilon > 0$, unless $NP = P$, there is no polynomial-time algorithm that approximates $\text{MAX\_INDEPENDENT\_SET}$ within a factor of $n^{\frac{1}{2} - \epsilon}$.

In fact, he showed something potentially stronger

**Theorem**
For any $\epsilon > 0$, unless $NP = ZPP$ there is no polynomial-time algorithm that approximates $\text{MAX\_INDEPENDENT\_SET}$ within a factor of $n^{1-\epsilon}$.

But a graph can have an independent set of size $cn$, so if $\frac{\text{opt}}{\text{approx}} > n^{1-\epsilon}$ then $\text{approx} < \frac{cn}{n^{1-\epsilon}} = cn^{\epsilon} \rightarrow c$ as $n \rightarrow \infty$
Johan Håstad [1999] proved

**Theorem**
For any \( \epsilon > 0 \), unless \( \text{NP} = \text{P} \), there is no polynomial-time algorithm that approximates \text{MAX\_INDEPENDENT\_SET} within a factor of \( n^{1/2 - \epsilon} \).

In fact, he showed something potentially stronger

**Theorem**
For any \( \epsilon > 0 \), unless \( \text{NP} = \text{ZPP} \) there is no polynomial-time algorithm that approximates \text{MAX\_INDEPENDENT\_SET} within a factor of \( n^{1-\epsilon} \).

But a graph can have an independent set of size \( cn \), so if \( \frac{\text{opt}}{\text{approx}} > n^{1-\epsilon} \) then \( \text{approx} < \frac{cn}{n^{1-\epsilon}} = cn^\epsilon \rightarrow c \) as \( n \rightarrow \infty \)

So this problem is generally considered to be **inapproximable**
Hardness of Approximation and Other Issues

But wait:

\[
\text{MAX INDEPENDENT SET} \leq p \text{VERTEX COVER}
\]

\[
\text{VERTEX COVER has a 2-approximation}
\]

WHAT'S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of MAX INDEPENDENT SET for which the optimal solution is \( n/2 \)

This transforms into an instance of VERTEX_COVER which the optimal solution is \( n/2 \)

The 2-approximation for VERTEX_COVER might return an approximation of size 2 \( (n/2) = n \)

This transforms back into an independent set of size 0! [A BAD APPROXIMATION!]

Hardness of Approximation and Other Issues

But wait:

- \text{MAX\_INDEPENDENT\_SET} \leq_{p} \text{VERTEX\_COVER}

What's going on?!

Problem transformation may not preserve approximation quality!

Consider an instance of \text{MAX\_INDEPENDENT\_SET} for which the optimal solution is \( \frac{n}{2} \).

This transforms into an instance of \text{VERTEX\_COVER} which the optimal solution is also \( \frac{n}{2} \).

The 2-approximation for \text{VERTEX\_COVER} might return an approximation of size \( 2 \times \frac{n}{2} = n \).

This transforms back into an independent set of size 0! (A bad approximation!)
Hardness of Approximation and Other Issues

But wait:

- \( \text{MAX\_INDEPENDENT\_SET} \leq_p \text{VERTEX\_COVER} \)
- \( \text{VERTEX\_COVER} \) has a 2-approximation

WHAT'S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of \( \text{MAX\_INDEPENDENT\_SET} \) for which the optimal solution is \( \frac{n}{2} \).

This transforms into an instance of \( \text{VERTEX\_COVER} \) which the optimal solution is \( \frac{n}{2} \).

The 2-approximation for \( \text{VERTEX\_COVER} \) might return an approximation of size 2 \( \left( \frac{n}{2} \right) = n \).

This transforms back into an independent set of size 0! [A BAD APPROXIMATION!]
Hardness of Approximation and Other Issues

But wait:

- \textsc{Max Independent Set} \leq_p \textsc{Vertex Cover}
- \textsc{Vertex Cover} has a 2-approximation

WHAT'S GOING ON?!

Problem transformation may not preserve approximation quality!
Consider an instance of \textsc{Max Independent Set} for which the optimal solution is $n/2$.
This transforms into an instance of \textsc{Vertex Cover} which the optimal solution is also $n/2$.
The 2-approximation for \textsc{Vertex Cover} might return an approximation of size $2(n/2) = n$.
This transforms back into an independent set of size 0! [A BAD APPROXIMATION!]
Hardness of Approximation and Other Issues

But wait:

- \( \text{MAX\_INDEPENDENT\_SET} \leq_p \text{VERTEX\_COVER} \)
- \( \text{VERTEX\_COVER} \) has a 2-approximation

WHAT’S GOING ON?!

Problem transformation may not preserve approximation quality!
Hardness of Approximation and Other Issues

But wait:

- \( \text{MAX}_\text{INDEPENDENT}_\text{SET} \leq_p \text{VERTEX}_\text{COVER} \)
- \( \text{VERTEX COVER} \) has a 2-approximation

WHAT’S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of \( \text{MAX}_\text{INDEPENDENT}_\text{SET} \) for which the optimal solution is \( n/2 \)
Hardness of Approximation and Other Issues

But wait:

- \text{MAX\_INDEPENDENT\_SET} \leq_p \text{VERTEX\_COVER}
- \text{VERTEX COVER} has a 2-approximation

WHAT’S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of \text{MAX\_INDEPENDENT\_SET} for which the optimal solution is \(n/2\)

- This transforms into an instance of \text{VERTEX\_COVER} which the optimal solution is \(n/2\)
Hardness of Approximation and Other Issues

But wait:

- MAX_INDEPENDENT_SET $\leq_p$ VERTEX_COVER
- VERTEX COVER has a 2-approximation

WHAT’S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of MAX_INDEPENDENT_SET for which the optimal solution is $n/2$

- This transforms into an instance of VERTEX_COVER which the optimal solution is $n/2$
- The 2-approximation for VERTEX_COVER might return an approximation of size $2(n/2) = n$
Hardness of Approximation and Other Issues

But wait:

- \( \text{MAX\_INDEPENDENT\_SET} \leq_p \text{VERTEX\_COVER} \)
- \( \text{VERTEX COVER} \) has a 2-approximation

WHAT’S GOING ON?!

Problem transformation may not preserve approximation quality!

Consider an instance of \( \text{MAX\_INDEPENDENT\_SET} \) for which the optimal solution is \( n/2 \)

- This transforms into an instance of \( \text{VERTEX\_COVER} \) which the optimal solution is \( n/2 \)
- The 2-approximation for \( \text{VERTEX\_COVER} \) might return an approximation of size \( 2(n/2) = n \)
- This transforms back into an independent set of size 0! [A BAD APPROXIMATION!]