Binary Knapsack

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

March 17, 2014

Outline

Outline

Review

Topics covered previously

Review

Topics covered previously

1. Problem definition.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.
3. The greedy algorithm for the fractional case.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.
3. The greedy algorithm for the fractional case.
4. The failure of the greedy algorithm for the binary case.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.
3. The greedy algorithm for the fractional case.
4. The failure of the greedy algorithm for the binary case.
5. A weight-based, pseudo-polynomial dynamic programming algorithm for exact solution.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.
3. The greedy algorithm for the fractional case.
4. The failure of the greedy algorithm for the binary case.
5. A weight-based, pseudo-polynomial dynamic programming algorithm for exact solution.
6. A profit-based, pseudo-polynomial dynamic programming algorithm for exact solution.

Review

Topics covered previously

1. Problem definition.
2. The binary and fractional cases.
3. The greedy algorithm for the fractional case.
4. The failure of the greedy algorithm for the binary case.
5. A weight-based, pseudo-polynomial dynamic programming algorithm for exact solution.
6. A profit-based, pseudo-polynomial dynamic programming algorithm for exact solution.
7. Scaling the profit-based pseudo-polynomial algorithm to develop a $(1-\varepsilon)$ factor approximation algorithm.

A greedy algorithm for binary knapsack

Greedy Algorithm

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal.

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it?

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1, p_1 = (1+\varepsilon), w_2 = 1, p_2 = (1+\varepsilon), \ldots w_{n-1} = 1, p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1$, $p_1 = (1+\varepsilon)$, $w_2 = 1$, $p_2 = (1+\varepsilon)$, $\ldots$ $w_{n-1} = 1$, $p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.
Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1) \cdot (1+\varepsilon)$*.*

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \dots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1$, $p_1 = (1+\varepsilon)$, $w_2 = 1$, $p_2 = (1+\varepsilon)$, $\dots$ $w_{n-1} = 1$, $p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.

Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1)\cdot(1+\varepsilon)$*. Optimal solution is* $k \cdot n$*, obtained by picking the* n^{th} *object.*

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1, p_1 = (1+\varepsilon), w_2 = 1, p_2 = (1+\varepsilon), \ldots w_{n-1} = 1, p_{n-1} = (1+\varepsilon),$ $w_n = p_n = W = k \cdot n.$
Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1)\cdot(1+\varepsilon)$*. Optimal solution is* $k \cdot n$*, obtained by picking the* n^{th} *object. Competitive ratio is*

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \dots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1, p_1 = (1+\varepsilon), w_2 = 1, p_2 = (1+\varepsilon), \dots w_{n-1} = 1, p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.
Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1)\cdot(1+\varepsilon)$. *Optimal solution is* $k \cdot n$*, obtained by picking the* n^{th} *object. Competitive ratio is* $\frac{k \cdot n}{(n-1)\cdot(1+\varepsilon)}$

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1$, $p_1 = (1+\varepsilon)$, $w_2 = 1$, $p_2 = (1+\varepsilon)$, $\ldots$ $w_{n-1} = 1$, $p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.
Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1)\cdot(1+\varepsilon)$*. Optimal solution is* $k \cdot n$*, obtained by picking the* n^{th} *object. Competitive ratio is* $\frac{k \cdot n}{(n-1)\cdot(1+\varepsilon)} \geq \frac{k}{1+\varepsilon}$,

A greedy algorithm for binary knapsack

Greedy Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint.

Note

As explained previously, the greedy algorithm is not optimal. How bad is it? Knapsack of capacity B. $w_1 = 1$, $p_1 = (1+\varepsilon)$, $w_2 = 1$, $p_2 = (1+\varepsilon)$, $\ldots$ $w_{n-1} = 1$, $p_{n-1} = (1+\varepsilon)$, $w_n = p_n = W = k \cdot n$.
Greedy solution will pack $(n-1)$ *objects of weight* 1*, for a total profit of* $(n-1)\cdot(1+\varepsilon)$*. Optimal solution is* $k \cdot n$*, obtained by picking the* n^{th} *object. Competitive ratio is* $\frac{k \cdot n}{(n-1)\cdot(1+\varepsilon)} \geq \frac{k}{1+\varepsilon}$*, i.e., unbounded.*

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \dots \frac{p_n}{w_n}$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

Let $\sum_{j=1}^{i-1} w_j = S$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

Let $\sum_{j=1}^{i-1} w_j = S$. We must have $S < W$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

Let $\sum_{j=1}^{i-1} w_j = S$. We must have $S < W$.
Note that $\sum_{j=1}^{i-1} p_j + \frac{W-S}{w_i} \cdot p_i \geq OPT$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

Let $\sum_{j=1}^{i-1} w_j = S$. We must have $S < W$.
Note that $\sum_{j=1}^{i-1} p_j + \frac{W-S}{w_i} \cdot p_i \geq OPT$.
Hence, $\sum_{j=1}^{i-1} p_j + p_i \geq OPT$.

A $\frac{1}{2}$-approximation algorithm

Bounded-error Algorithm

1. Order the objects so that $\frac{p_1}{w_1} \geq \frac{p_2}{w_3} \geq \ldots \frac{p_n}{w_n}$.
2. Pack objects into knapsack, till you reach an object which cannot be packed without violating the weight constraint. Let this object be o_i.
3. Pick the better of $\{o_1, o_2, \ldots, o_{i-1}\}$ and $\{o_i\}$.

Theorem

The above algorithm is a $\frac{1}{2}$-approximation algorithm.

Proof.

Let $\sum_{j=1}^{i-1} w_j = S$. We must have $S < W$.
Note that $\sum_{j=1}^{i-1} p_j + \frac{W-S}{w_i} \cdot p_i \geq OPT$.
Hence, $\sum_{j=1}^{i-1} p_j + p_i \geq OPT$. It follows that $\max\{\sum_{j=1}^{i-1} p_j, p_i\} \geq \frac{OPT}{2}$. □

Recalling the FPTAS for knapsack

Recalling the FPTAS for knapsack

The scheme

Recalling the FPTAS for knapsack

The scheme

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p_i' = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.
   Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.
   Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ϵ, compute $K = \frac{\epsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.
   Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that,

Recalling the FPTAS for knapsack

The scheme

1. Given ϵ, compute $K = \frac{\epsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.
   Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p'_i \leq \sum_{o_i \in S'} p_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p'_i \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p'_i$.

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p_i' = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p_i' \leq p_i$. However, $p_i \leq K \cdot p_i' + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p_i' \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p_i' \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p_i' \geq \sum_{o_i \in O} p_i'$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p_i' \geq K \cdot \sum_{o_i \in O} p_i'$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p_i' \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p_i'$. We conclude that, $\sum_{o_i \in S'} p_i \geq \sum_{o_i \in O} p_i - n \cdot K$.

Recalling the FPTAS for knapsack

The scheme

1. Given ϵ, compute $K = \frac{\epsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p'_i \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p'_i$. We conclude that, $\sum_{o_i \in S'} p_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
6. Finally, observe that $\sum_{o_i \in O} p_i = OPT$ and that $n \cdot K = \epsilon \cdot P \leq \epsilon \cdot OPT$, i.e.,

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p_i' = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p_i' \leq p_i$. However, $p_i \leq K \cdot p_i' + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p_i' \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p_i' \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p_i' \geq \sum_{o_i \in O} p_i'$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p_i' \geq K \cdot \sum_{o_i \in O} p_i'$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p_i' \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p_i'$. We conclude that, $\sum_{o_i \in S'} p_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
6. Finally, observe that $\sum_{o_i \in O} p_i = OPT$ and that $n \cdot K = \varepsilon \cdot P \leq \varepsilon \cdot OPT$, i.e.,$\sum_{o_i \in S'} p_i \geq$

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$. Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p'_i \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p'_i$. We conclude that, $\sum_{o_i \in S'} p_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
6. Finally, observe that $\sum_{o_i \in O} p_i = OPT$ and that $n \cdot K = \varepsilon \cdot P \leq \varepsilon \cdot OPT$, i.e.,$\sum_{o_i \in S'} p_i \geq OPT - \varepsilon \cdot OPT$

Recalling the FPTAS for knapsack

The scheme

1. Given ε, compute $K = \frac{\varepsilon \cdot P}{n}$, where $P = \max_i p_i$.
2. For each object o_i, define $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
3. Use dynamic programming to compute the optimal set S' of the truncated instance.
4. Output S'.

Analysis

1. Because of rounding down, $K \cdot p'_i \leq p_i$. However, $p_i \leq K \cdot p'_i + K$.
2. Let O denote the optimal set. It follows that $\sum_{o_i \in O} p_i - K \cdot \sum_{o_i \in O} p'_i \leq n \cdot K$.
   Hence, $K \cdot \sum_{o_i \in O} p'_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
3. However, S' is the optimal set under profit assignment p'. Therefore, $\sum_{o_i \in S'} p'_i \geq \sum_{o_i \in O} p'_i$.
4. It follows that, $K \cdot \sum_{o_i \in S'} p'_i \geq K \cdot \sum_{o_i \in O} p'_i$.
5. Observe that, $K \cdot \sum_{o_i \in S'} p'_i \leq \sum_{o_i \in S'} p_i$. Hence, $\sum_{o_i \in S'} p_i \geq K \cdot \sum_{o_i \in O} p'_i$. We conclude that,
   $\sum_{o_i \in S'} p_i \geq \sum_{o_i \in O} p_i - n \cdot K$.
6. Finally, observe that $\sum_{o_i \in O} p_i = OPT$ and that $n \cdot K = \varepsilon \cdot P \leq \varepsilon \cdot OPT$,
   i.e.,$\sum_{o_i \in S'} p_i \geq OPT - \varepsilon \cdot OPT = (1 - \varepsilon) \cdot OPT$!

Strong NP-hardness and Approximation schemes

Strong NP-hardness and Approximation schemes

Chief points

Strong NP-hardness and Approximation schemes

Chief points

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input.

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs)

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight).

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.
4. A problem is said to be strongly **NP-hard**, if it is **NP-hard** in the unary sense,

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.
4. A problem is said to be strongly **NP-hard**, if it is **NP-hard** in the unary sense, i.e., numbers do not matter.

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.
4. A problem is said to be strongly **NP-hard**, if it is **NP-hard** in the unary sense, i.e., numbers do not matter.

Theorem

A strongly **NP-hard** *problem cannot have a pseudo-polynomial algorithm, unless* **P=NP**.

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.
4. A problem is said to be strongly **NP-hard**, if it is **NP-hard** in the unary sense, i.e., numbers do not matter.

Theorem

A strongly **NP-hard** *problem cannot have a pseudo-polynomial algorithm, unless* **P=NP**.

Theorem

Let Π denote an **NP-hard** *minimization problem. Assume, for all instances I, $OPT(I) < p(|I_u|)$.*

Strong NP-hardness and Approximation schemes

Chief points

1. The notion of size of input. Objects (sets, graphs) and numbers (cost, weight). Measuring input size in unary ($|I_u|$) and binary ($|I|$).
2. Making an algorithm's performance better by measuring in unary.
3. An algorithm for a problem Π. is *pseudo-polynomial*, if it runs in time polynomial in $|I_u|$, for all instances, I of the problem.
4. A problem is said to be strongly **NP-hard**, if it is **NP-hard** in the unary sense, i.e., numbers do not matter.

Theorem

A strongly **NP-hard** *problem cannot have a pseudo-polynomial algorithm, unless* **P=NP***.*

Theorem

Let Π denote an **NP-hard** *minimization problem. Assume, for all instances I, $OPT(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudo-polynomial time algorithm.*

Strong NP-hardness

Strong NP-hardness

Proof.

Strong NP-hardness

Proof.

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$A(I) \leq$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$A(I) \quad \leq \quad (1+\varepsilon)OPT(I)$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq \end{aligned}$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \end{aligned}$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< \end{aligned}$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< OPT(I) + \varepsilon \cdot p(|I_u|) \end{aligned}$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$
\begin{aligned}
A(I) &\leq (1+\varepsilon)OPT(I) \\
&\leq OPT(I) + \varepsilon \cdot OPT(I) \\
&< OPT(I) + \varepsilon \cdot p(|I_u|) \\
&=
\end{aligned}
$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< OPT(I) + \varepsilon \cdot p(|I_u|) \\ &= OPT(I) + 1 \end{aligned}$$

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< OPT(I) + \varepsilon \cdot p(|I_u|) \\ &= OPT(I) + 1 \end{aligned}$$

The FPTAS is now forced to produce the optimal answer!

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< OPT(I) + \varepsilon \cdot p(|I_u|) \\ &= OPT(I) + 1 \end{aligned}$$

The FPTAS is now forced to produce the optimal answer! Running time is $q(|I|, p(|I_u|)$, which is polynomial in $|I_u|$,

Strong NP-hardness

Proof.

1. Assume that Π admits an FPTAS running in time $q(|I|, \frac{1}{\varepsilon})$, in inputs I and ε, where q is some polynomial.
2. Set $\varepsilon = \frac{1}{p(|I_u|)}$ and run the FPTAS.
3. Observe that,

$$\begin{aligned} A(I) &\leq (1+\varepsilon)OPT(I) \\ &\leq OPT(I) + \varepsilon \cdot OPT(I) \\ &< OPT(I) + \varepsilon \cdot p(|I_u|) \\ &= OPT(I) + 1 \end{aligned}$$

The FPTAS is now forced to produce the optimal answer! Running time is $q(|I|, p(|I_u|)$, which is polynomial in $|I_u|$, i.e., we now have a pseudo-polynomial time algorithm for Π. □

Corollary

If Π is an **NP-hard** *minimization problem, as constrained above, then Π does not admit an FPTAS, assuming* **P** $\neq$ **NP**.