Aalto University
School of Science

Department of Computer Science

Combinatorics of Efficient Computations

Approximation Algorithms

Lecture 8: FPTAS for Knapsack via Scaling

Joachim Spoerhase

2019

Approximation Scheme

Let Π be an optimization problem. An algorithm $\mathcal{A}$ is called **polynomial time approximation scheme** (PTAS), if it computes for every (I, ϵ) with $I \in D_\Pi$ and $\epsilon > 0$ a solution $s \in S_\Pi(I)$ with the following properties:

- $\text{obj}_\Pi(I, s) \leq (1 + \epsilon) \cdot \text{OPT}$, if Π is a minimization problem,
- $\text{obj}_\Pi(I, s) \geq (1 - \epsilon) \cdot \text{OPT}$, if Π is a maximization problem.

The running time of $\mathcal{A}$ is polynomial in $|I|$ for every fixed $\epsilon > 0$.

Approximation Scheme

Let Π be an optimization problem. An algorithm $\mathcal{A}$ is called **polynomial time approximation scheme** (PTAS), if it computes for every (I, ϵ) with $I \in D_\Pi$ and $\epsilon > 0$ a solution $s \in S_\Pi(I)$ with the following properties:

- $\text{obj}_\Pi(I, s) \leq (1 + \epsilon) \cdot \text{OPT}$, if Π is a minimization problem,
- $\text{obj}_\Pi(I, s) \geq (1 - \epsilon) \cdot \text{OPT}$, if Π is a maximization problem.

The running time of $\mathcal{A}$ is polynomial in $|I|$ for every fixed $\epsilon > 0$.

$\mathcal{A}$ is called **fully polynomial time approximation scheme** (FPTAS), if its running time is polynomial in $|I|$ and $1/\epsilon$.

Approximation Scheme

Let Π be an optimization problem. An algorithm $\mathcal{A}$ is called **polynomial time approximation scheme** (PTAS), if it computes for every (I, ϵ) with $I \in D_\Pi$ and $\epsilon > 0$ a solution $s \in S_\Pi(I)$ with the following properties:

- $\text{obj}_\Pi(I, s) \le (1 + \epsilon) \cdot \text{OPT}$, if Π is a minimization problem,
- $\text{obj}_\Pi(I, s) \ge (1 - \epsilon) \cdot \text{OPT}$, if Π is a maximization problem.

The running time of $\mathcal{A}$ is polynomial in $|I|$ for every fixed $\epsilon > 0$.

$\mathcal{A}$ is called **fully polynomial time approximation scheme** (FPTAS), if its running time is polynomial in $|I|$ and $1/\epsilon$.

Example running times

- $O(n^{1/\epsilon}) \rightsquigarrow$ polynomial time approximation scheme
- $O(2^{1/\epsilon} n^4) \rightsquigarrow$ polynomial time approximation scheme
- $O(n^3/\epsilon^2) \rightsquigarrow$ fully polynomial time approximation scheme (FPTAS)

Knapsack Problem

We are given a set $S = \{a_1, \ldots, a_n\}$ of **objects**. For every object a_i, $i = 1, \ldots, n$ two quantities size$(a_i) \in \mathbb{N}^+$ and profit$(a_i) \in \mathbb{N}^+$ are specified. Moreover, we are given a knapsack **capacity** $B \in \mathbb{N}^+$. We are looking for a subset of objects whose total size is at most B and whose total profit is maximized.

Knapsack Problem

We are given a set $S = \{a_1, \ldots, a_n\}$ of **objects**. For every object a_i, $i = 1, \ldots, n$ two quantities size$(a_i) \in \mathbb{N}^+$ and profit$(a_i) \in \mathbb{N}^+$ are specified. Moreover, we are given a knapsack **capacity** $B \in \mathbb{N}^+$. We are looking for a subset of objects whose total size is at most B and whose total profit is maximized.

NP-hard

Pseudopolynomial Algorithm

Let Π be an optimization problem whose instances are specified by discrete **objects** (for example sets, graphs, or strings) and **numbers** (such as costs, weights, profits). By $|I|$ we denote (as usual) the size of the instance $I \in D_{\Pi}$ where all numbers in I are encoded in **binary**. By $|I_{\mathsf{u}}|$ we denote the size of I when all numbers in I are encoded in **unary**.

Pseudopolynomial Algorithm

Let Π be an optimization problem whose instances are specified by discrete **objects** (for example sets, graphs, or strings) and **numbers** (such as costs, weights, profits). By $|I|$ we denote (as usual) the size of the instance $I \in D_\Pi$ where all numbers in I are encoded in **binary**. By $|I_\mathsf{u}|$ we denote the size of I when all numbers in I are encoded in **unary**.

- The running time of a polynomial algorithm for Π is polynomial in $|I|$.

Pseudopolynomial Algorithm

Let Π be an optimization problem whose instances are specified by discrete **objects** (for example sets, graphs, or strings) and **numbers** (such as costs, weights, profits). By $|I|$ we denote (as usual) the size of the instance $I \in D_\Pi$ where all numbers in I are encoded in **binary**. By $|I_u|$ we denote the size of I when all numbers in I are encoded in **unary**.

- The running time of a polynomial algorithm for Π is polynomial in $|I|$.
- The running time of a **pseudo-polynomial algorithm** is polynomial in $|I_u|$

Pseudopolynomial Algorithm

Let Π be an optimization problem whose instances are specified by discrete **objects** (for example sets, graphs, or strings) and **numbers** (such as costs, weights, profits). By $|I|$ we denote (as usual) the size of the instance $I \in D_{\Pi}$ where all numbers in I are encoded in **binary**. By $|I_{\mathsf{u}}|$ we denote the size of I when all numbers in I are encoded in **unary**.

- The running time of a polynomial algorithm for Π is polynomial in $|I|$.
- The running time of a **pseudo-polynomial algorithm** is polynomial in $|I_{\mathsf{u}}|$
- The running time of a pseudo-polynomial algorithm is not always polynomial in $|I|$

Pseudopolynomial algorithm for KNAPSACK

- $P := \max_i \mathsf{profit}(a_i) \rightsquigarrow \mathsf{OPT} \leq nP$

Pseudopolynomial algorithm for Knapsack

- $P := \max_i \text{profit}(a_i) \rightsquigarrow \text{OPT} \leq nP$

- For every $i = 1, \ldots, n$ and every $p \in \{1, \ldots, nP\}$ let $S_{i,p}$ be a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimum among all subsets with these properties.

Pseudopolynomial algorithm for KNAPSACK

- $P := \max_i \text{profit}(a_i) \rightsquigarrow \text{OPT} \leq nP$

- For every $i = 1, \ldots, n$ and every $p \in \{1, \ldots, nP\}$ let $S_{i,p}$ be a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimum among all subsets with these properties.

- $A(i, p)$ denotes the total size of the set $S_{i,p}$ (we set $A(i, p) = \infty$ if such a set does not exist).

Pseudopolynomial algorithm for KNAPSACK

- $P := \max_i \text{profit}(a_i) \rightsquigarrow \text{OPT} \leq nP$

- For every $i = 1, \ldots, n$ and every $p \in \{1, \ldots, nP\}$ let $S_{i,p}$ be a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimum among all subsets with these properties.

- $A(i,p)$ denotes the total size of the set $S_{i,p}$ (we set $A(i,p) = \infty$ if such a set does not exist).

- If all $A(i,p)$ are known then OPT can be determined by $\max\{\, p \mid A(n,p) \leq B \,\}$

Pseudo-Polynomial Algorithm for $\textsc{Knapsack}$

- $A(1, p)$ is known for all $p \in \{0, \ldots, nP\}$

Pseudo-Polynomial Algorithm for Knapsack

- $A(1, p)$ is known for all $p \in \{0, \ldots, nP\}$
- We set $A(i, p) := \infty$ for $p < 0$

Pseudo-Polynomial Algorithm for Knapsack

- $A(1, p)$ is known for all $p \in \{0, \ldots, nP\}$

- We set $A(i, p) := \infty$ for $p < 0$

- $A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1})\}$

Pseudo-Polynomial Algorithm for Knapsack

- $A(1, p)$ is known for all $p \in \{0, \ldots, nP\}$

- We set $A(i, p) := \infty$ for $p < 0$

- $A(i+1, p) = \min\{A(i, p), \mathsf{size}(a_{i+1}) + A(i, p - \mathsf{profit}(a_{i+1})\}$

- $\leadsto$ All values $A(i, p)$ and therefore OPT can be computed in $O(n^2 P)$ time

Pseudo-Polynomial Algorithm for Knapsack

- $A(1, p)$ is known for all $p \in \{0, \ldots, nP\}$

- We set $A(i, p) := \infty$ for $p < 0$

- $A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}$

- $\rightsquigarrow$ All values $A(i, p)$ and therefore OPT can be computed in $O(n^2 P)$ time

Knapsack can be solved in pseudo-polynomial time $O(n^2 P)$.

FPTAS for Knapsack via Scaling

- Running time $O(n^2 P)$ polynomial in n, if P is polynomial in n

FPTAS for KNAPSACK via Scaling

- Running time $O(n^2P)$ polynomial in n, if P is polynomial in n
- FPTAS idea: **Scale** profits to polynomial size (depending on the required error parameter ϵ).

FPTAS for KNAPSACK via Scaling

KnapsackFPTAS(I,ϵ)

$K \leftarrow \frac{\epsilon P}{n}$

$\text{profit}'(a_i) := \left\lfloor \frac{\text{profit}(a_i)}{K} \right\rfloor$

compute optimum solution S' for I with respect to $\text{profit}'(\cdot)$

return S'

FPTAS for Knapsack via Scaling

KnapsackFPTAS(I,ϵ)

$K \leftarrow \frac{\epsilon P}{n}$

$\text{profit}'(a_i) := \left\lfloor \frac{\text{profit}(a_i)}{K} \right\rfloor$

compute optimum solution S' for I with respect to $\text{profit}'(\cdot)$

return S'

Lemma The solution S' satisfies $\text{profit}(S') \geq (1 - \epsilon) \cdot \text{OPT}$.

FPTAS for KNAPSACK via Scaling

KnapsackFPTAS(I,ϵ)

$K \leftarrow \frac{\epsilon P}{n}$

$\text{profit}'(a_i) := \left\lfloor \frac{\text{profit}(a_i)}{K} \right\rfloor$

compute optimum solution S' for I with respect to $\text{profit}'(\cdot)$

return S'

Lemma The solution S' satisfies $\text{profit}(S') \geq (1 - \epsilon) \cdot \text{OPT}$.

Theorem KnapsackFPTAS is an FPTAS for KNAPSACK with running time $O(n^3/\epsilon)$.

Strong NP-Hardness

An optimization problem is **strongly NP-hard**, if it remains NP-hard also with unary numbers.

Theorem A strongly NP-hard problem has no pseudo-polynomial algorithm unless $\mathsf{P} = \mathsf{NP}$.

FPTAS and Pseudo-Polynomial Algorithms

Theorem Let p be a polynomial. Let Π be an NP-hard minimization problem with integer objective function and with $\mathrm{OPT}(I) < p(|I_{\mathsf{u}}|)$ for all instances I of Π. If Π admits an FPTAS then there is also a pseudo-polynomial algorithm for Π.

FPTAS und Strong NP-Hardness

Corollary Let Π be an NP-hard optimization problem, that satisfies the requirements of the previous theorem. If Π is strongly NP-hard then there is no FPTAS for Π unless $P = NP$.