A Faster FPTAS for #Knapsack

Paweł Gawrychowski $^1$ Liran Markin $^2$ Oren Weimann $^2$

$^1$University of Wrocław, Poland

$^2$University of Haifa, Israel

ICALP 2018

Slides by Liran Markin
Counting Knapsack Solutions

given a set $W = \{w_1, w_2, ..., w_n\}$ of $n$ non-negative integers and a capacity $C$, count the number of subsets of $W$ with total sum of at most $C$. 

Gawrychowski, Markin, Weimann (University of Wrocław, Poland, University of Haifa, Israel)

A Faster FPTAS for #Knapsack

ICALP 2018
Counting Knapsack Solutions

Given a set \( W = \{w_1, w_2, \ldots, w_n\} \) of \( n \) non-negative integers and a capacity \( C \), count the number of subsets of \( W \) with total sum of at most \( C \).
Given a set \( W = \{w_1, w_2, \ldots, w_n\} \) of \( n \) non-negative integers and a capacity \( C \), count the number of subsets of \( W \) with total sum of at most \( C \).
Naïve Algorithm

Recurse on the last item.
Naïve Algorithm

Recurse on the last item. A solution from the set $W$ and capacity $C$ can be obtained by
Naïve Algorithm

Recurse on the last item. A solution from the set $W$ and capacity $C$ can be obtained by

- leaving the last element $w_n$, and taking a solution from $W/\{w_n\}$ and capacity $C$. 
Naïve Algorithm

Recurse on the last item. A solution from the set $W$ and capacity $C$ can be obtained by

- leaving the last element $w_n$, and taking a solution from $W/\{w_n\}$ and capacity $C$.
- taking the last element $w_n$, and taking the rest of the elements from $W/\{w_n\}$ such that the capacity is $C - w_n$. 

$$\text{f}(n, C) = \text{f}(n-1, C) + \text{f}(n-1, C - w_n)$$

$O(nC)$ time but $C$ is large!
Naïve Algorithm

Recurse on the last item. A solution from the set $W$ and capacity $C$ can be obtained by

- leaving the last element $w_n$, and taking a solution from $W/\{w_n\}$ and capacity $C$.
- taking the last element $w_n$, and taking the rest of the elements from $W/\{w_n\}$ such that the capacity is $C - w_n$.

$$f(n, C) = f(n - 1, C) + f(n - 1, C - w_n)$$
Naïve Algorithm

Recurse on the last item. A solution from the set $W$ and capacity $C$ can be obtained by

- leaving the last element $w_n$, and taking a solution from $W/\{w_n\}$ and capacity $C$.
- taking the last element $w_n$, and taking the rest of the elements from $W/\{w_n\}$ such that the capacity is $C - w_n$.

$$f(n, C) = f(n - 1, C) + f(n - 1, C - w_n)$$

$O(nC)$ time but $C$ is large!
**Fully Polynomial Time Approximation Scheme (FPTAS)**

**Definition**

Given $\varepsilon > 0$, estimate the number of solutions with ratio $(1 \pm \varepsilon)$, and run in polynomial time in the size of the input and in $1/\varepsilon$. 

**Deterministic FPTAS**

$O(n^{3\varepsilon - 1}\log(n\varepsilon^{-1}))/\varepsilon$ 

[Štefanković et al. 2012], [Gopalan et al. 2011].

**Best deterministic FPTAS**

$O(n^{3\varepsilon - 1}\log(\varepsilon^{-1}/\log n))$  

[Rizzi, Tomescu 2014].

**Best randomized FPTAS**

$O(n^{2.5\sqrt{\log(n\varepsilon^{-1})} + \varepsilon^{-2}})$  

[Dyer 2003].

This work

A deterministic FPTAS running in $O(n^{2.5\varepsilon - 1.5\log(n\varepsilon^{-1})})$ time and $O(n^{1.5\varepsilon - 1.5})$ space.
Fully Polynomial Time Approximation Scheme (FPTAS)

Definition
Given $\varepsilon > 0$, estimate the number of solutions with ratio $(1 \pm \varepsilon)$, and run in polynomial time in the size of the input and in $1/\varepsilon$.

Deterministic FPTAS $O(n^3\varepsilon^{-1} \log(n\varepsilon^{-1}))$ [Štefankovič et al. 2012], [Gopalan et al. 2011].
Best deterministic FPTAS $O(n^3\varepsilon^{-1} \log \varepsilon^{-1} / \log n)$ [Rizzi, Tomescu 2014].
Best randomized FPTAS $O(n^{2.5} \sqrt{\log(n\varepsilon^{-1})} + \varepsilon^{-2} n^2)$ [Dyer 2003].
Fully Polynomial Time Approximation Scheme (FPTAS)

**Definition**

Given $\varepsilon > 0$, estimate the number of solutions with ratio $(1 \pm \varepsilon)$, and run in polynomial time in the size of the input and in $1/\varepsilon$.

Deterministic FPTAS $O(n^3 \varepsilon^{-1} \log(n \varepsilon^{-1}))$ [Štefankovič et al. 2012], [Gopalan et al. 2011].

Best deterministic FPTAS $O(n^3 \varepsilon^{-1} \log \varepsilon^{-1} / \log n)$ [Rizzi, Tomescu 2014].

Best randomized FPTAS $O(n^{2.5} \sqrt{\log(n \varepsilon^{-1})} + \varepsilon^{-2} n^2)$ [Dyer 2003].

**This work**

A deterministic FPTAS running in $O(n^{2.5} \varepsilon^{-1.5} \log(n \varepsilon^{-1}) \log(n \varepsilon))$ time and $O(n^{1.5} \varepsilon^{-1.5})$ space.
Sum Approximation

Same idea as K-approximation sets [Halman 2009].
Sum Approximation

Same idea as K-approximation sets [Halman 2009].

For a function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ (number of solutions by capacity):
Let $f^\leq(x) = \sum_{0 \leq y \leq x} f(y)$ be the partial sum of $f$. 
Sum Approximation

Same idea as K-approximation sets [Halman 2009].

For a function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ (number of solutions by capacity):
Let $f^\leq(x) = \sum_{0 \leq y \leq x} f(y)$ be the partial sum of $f$.

**Definition**

A function $F$ is a $(1 + \epsilon)$-sum approximation of $f$ if for every $x$,

$$f^\leq(x) \leq F^\leq(x) \leq (1 + \epsilon)f^\leq(x)$$

---

Gawrychowski, Markin, Weimann
A Faster FPTAS for #Knapsack
ICALP 2018 5 / 13
Sum Approximation - Sparsification

\[ f(x) \]

Gawrychowski, Markin, Weimann

A Faster FPTAS for #Knapsack
Sum Approximation - Sparsification

\[ f(x) \]

wider circle → larger value
Sum Approximation - Sparsification

\[ f^\leq(x) \]
Sum Approximation - Sparsification

\[ f(x) \leq (1 + \epsilon) r_i \]

\[ r_{i+1} = (1 + \epsilon) r_i \]

Gawrychowski, Markin, Weimann (University of Wrocław, Poland, University of Haifa, Israel)
A Faster FPTAS for #Knapsack
ICALP 2018 6 / 13
Sum Approximation - Sparsification

\[ f_{\leq}(x) \]

\[ r_{i+1} = (1 + \epsilon)r_i \]

Gawrychowski, Markin, Weimann (University of Wrocław, Poland, University of Haifa, Israel)

A Faster FPTAS for #Knapsack

ICALP 2018 6 / 13
Sum Approximation - Sparsification

\[ f \leq x \]

\[ r_{i+1} = (1 + \epsilon)r_i \]
$f^{\leq}(x)$
Sum Approximation - Sparsification

\[ \tilde{f}(x) \]

\[ x_1 \xrightarrow{\epsilon} x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x \]
Sum Approximation - Sparsification

\[ F(x) \]

\[
\sum_{i=1}^{n} x_i = (1 + \epsilon) \sum_{i=1}^{n} r_i
\]
Sum Approximation - Sparsification

\[ F^\leq(x) = \tilde{f}^\leq(x) \leq (1 + \epsilon)f^\leq(x) \]
Claim

The size of $F$ is $|F| = |r| = \log_{1+\epsilon}(M).

Where $M$ is the sum of all values of $f$. 
Claim
The size of $F$ is $|F| = |r| = \log_{1+\epsilon} M$.

Where $M$ is the sum of all values of $f$.

For #Knapsack:
$M \leq \#\text{subsets of } W = 2^n$

Claim
The size of $F$ is $|F| = n/\epsilon$
Lemma

Let $F, G$ be $(1 + \epsilon)$-sum approximation of $f, g$.

Approximation: A $(1 + \epsilon')$-sum approximation of $F$ is a $(1 + \epsilon')(1 + \epsilon)$-sum approximation of $f$.

Summation: $(F + G)$ is a $(1 + \epsilon)$-sum approximation of $(f + g)$.

Shifting: $F(x - w)$ is a $(1 + \epsilon)$-sum approximation of $f(x - w)$ for any $w > 0$. 
Lemma

Let $F, G$ be $(1 + \epsilon)$-sum approximation of $f, g$.

**Approximation:** A $(1 + \epsilon')$-sum approximation of $F$ is a $(1 + \epsilon')(1 + \epsilon)$-sum approximation of $f$.

**Summation:** $(F + G)$ is a $(1 + \epsilon)$-sum approximation of $(f + g)$.

**Shifting:** $F(x - w)$ is a $(1 + \epsilon)$-sum approximation of $f(x - w)$ for any $w > 0$.

**Convolution:** $(F \ast G)$ is a $(1 + \epsilon)^2$-sum approximation of $(f \ast g)$.
Definition

Let $k_S(x)$ be a function that equals to the number of subsets of the set $S$ with a total weight of \textbf{exactly} $x$.

The answer to the \#Knapsack instance is $k_{\leq W}^*(C)$. 
Let $k_S(x)$ be a function that equals to the number of subsets of the set $S$ with a total weight of *exactly* $x$.

The answer to the #Knapsack instance is $k_{\leq W}(C)$.

$K_S$ is the sum-approximation of $k_S$. 
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):

- Shift \( K_S \) by \( w \).
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):
- Shift \( K_S \) by \( w \).
- Sum \( K_S \) with \( K_S(x - w) \).
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):
- Shift \( K_S \) by \( w \).
- Sum \( K_S \) with \( K_S(x - w) \).
- Sparsify with parameter \( (1 + \epsilon)^{1/n} \).
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):

- Shift \( K_S \) by \( w \).
- Sum \( K_S \) with \( K_S(x - w) \).
- Sparsify with parameter \((1 + \epsilon)^{1/n}\).

\( n \) steps \( \cdot |K_S| \) time
The Previous Best Algorithm [Štefankovič et al. 2012], [Halman 2016]

As in the naive algorithm

\[ k_{S \cup \{w\}}(x) = k_S(x) + k_S(x - w) \]

For every item \( w \) in \( W \):

- Shift \( K_S \) by \( w \).
- Sum \( K_S \) with \( K_S(x - w) \).
- Sparsify with parameter \((1 + \epsilon)^{1/n}\).

\( n \) steps \( \cdot |K_S| \) time \( \Rightarrow O(n^3/\epsilon) \)
Our Algorithm

Key observation:

\[ k_{S\cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) \]
Our Algorithm

Key observation:

\[
k_{S \cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) = (k_S \ast k_T)(x)
\]
Our Algorithm

Key observation:

\[ k_{S \cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) = (k_S \ast k_T)(x) \]

The algorithm:

- If the size of \( W \) is less than \( \sqrt{n} \), use the previous algorithm.
Our Algorithm

Key observation:

\[ k_{S \cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) = (k_S \ast k_T)(x) \]

The algorithm:

- If the size of \( W \) is less than \( \sqrt{n} \), use the previous algorithm.
- Split the set \( W \) into two halves \( S \) and \( T \).
Our Algorithm

Key observation:

\[ k_{S \cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) = (k_S \ast k_T)(x) \]

The algorithm:
- If the size of \( W \) is less than \( \sqrt{n} \), use the previous algorithm.
- Split the set \( W \) into two halves \( S \) and \( T \).
- Compute the sum-approximations \( K_S \) and \( K_T \) recursively.
Our Algorithm

Key observation:

\[ k_{S \cup T}(x) = \sum_{y \leq x} k_S(y)k_T(x - y) = (k_S * k_T)(x) \]

The algorithm:

- If the size of \( W \) is less than \( \sqrt{n} \), use the previous algorithm.
- Split the set \( W \) into two halves \( S \) and \( T \).
- Compute the sum-approximations \( K_S \) and \( K_T \) recursively.
- Compute \( K_W = K_{S \cup T} \) by convolution of \( K_S \) and \( K_T \), then sparsify to keep size small.
Our Algorithm

Key observation:

\[ k_{S \cup T}(x) = \sum_{y \leq x} k_S(y) k_T(x - y) = (k_S \ast k_T)(x) \]

The algorithm:

- If the size of \( W \) is less than \( \sqrt{n} \), use the previous algorithm.
- Split the set \( W \) into two halves \( S \) and \( T \).
- Compute the sum-approximations \( K_S \) and \( K_T \) recursively.
- Compute \( K_W = K_{S \cup T} \) by convolution of \( K_S \) and \( K_T \), then sparsify to keep size small.

\[ O(n^{2.5} \varepsilon^{-1.5} \log(n\varepsilon^{-1}) \log(n\varepsilon)) \text{ time and } O(n^{1.5} \varepsilon^{-1.5}) \text{ space.} \]

Sparsification parameter should be adjusted for every level of the recursion. See the paper.
Counting Integer Knapsack Solutions

$w_1, u_1$

$w_2, u_2$

capacity $C$

There is a FPTAS running in $O\left(n^{2.5\epsilon - 1.5\log \left(n\epsilon - 1\log U\right)\log \left(n\epsilon\right)\log 2U}\right)$ time and $O\left(n^{1.5\epsilon - 1.5\log U}\right)$ space.
Counting Integer Knapsack Solutions

\[ w_1, u_1 \quad w_2, u_2 \quad \text{capacity } C \]

Best FPTAS \( O(n^3\epsilon^{-1} \log(n\epsilon^{-1} \log U) \log^2 U) \) [Halman 2016].
Counting Integer Knapsack Solutions

\[ w_1, u_1 \quad w_2, u_2 \quad \text{capacity } C \]

Best FPTAS \( O(n^3 \epsilon^{-1} \log(n \epsilon^{-1} \log U) \log^2 U) \) [Halman 2016].

There is a FPTAS running in \( O(n^{2.5} \epsilon^{-1.5} \log(n \epsilon^{-1} \log U) \log(n \epsilon) \log^2 U) \) time and \( O(n^{1.5} \epsilon^{-1.5} \log U) \) space.
Open Problems

- Ignore the dependency on $\epsilon$ (constant), is there an algorithm with running time of $\tilde{O}(n^{2.5-\alpha})$?
- Deterministic FPTAS with running time of $\tilde{O}(n^{2.5\epsilon^{-1.5+\alpha}})$?
Open Problems

- Ignore the dependency on $\epsilon$ (constant), is there an algorithm with running time of $\tilde{O}(n^{2.5-\alpha})$?
- Deterministic FPTAS with running time of $\tilde{O}(n^{2.5}\epsilon^{-1.5+\alpha})$?

Thank You!

Questions?