

Truthful Unification Framework for Packing Integer Programs with Choices

Iftah Gamzu

School of Computer Science, Tel Aviv University

Joint work with **Yossi Azar**, Microsoft Research and Tel Aviv University

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application

Outline

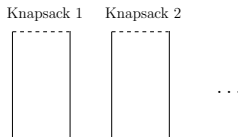
- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application
- 3 **Concluding Remarks**

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application
- 3 **Concluding Remarks**

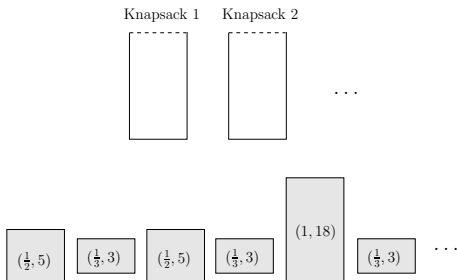
Multiple knapsack input

- A collection of unit-capacity **knapsacks**.



Multiple knapsack input

- A collection of unit-capacity **knapsacks**.
- A set of **items**, each characterized by a pair (s_i, v_i) .
 - s_i is the **size** of the item.
 - v_i is the **value** gained by placing the item in some knapsack.



Multiple knapsack input

- A collection of unit-capacity **knapsacks**.
- A set of **items**, each characterized by a pair (s_i, v_i) .
 - s_i is the **size** of the item.
 - v_i is the **value** gained by placing the item in some knapsack.

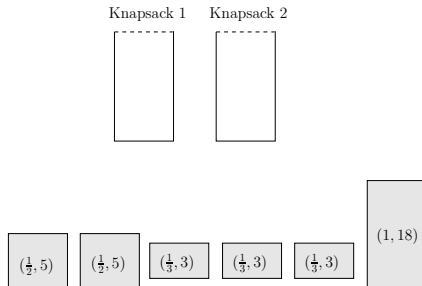
Multiple knapsack objective

Select a subset of the items (along with a knapsack for each selected item) such that

- all the items can be simultaneously placed in their designated knapsacks, while **respecting the capacities**.
- has a **maximum value**.

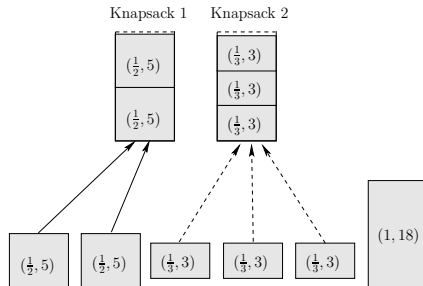
Example

- 2 unit-capacity knapsacks.
- A set of items $\left\{ \underbrace{\left(\frac{1}{2}, 5\right)}_{\times 2}, \underbrace{\left(\frac{1}{3}, 3\right)}_{\times 3}, \underbrace{(1, 18)}_{\times 1} \right\}$.



Example

- 2 unit-capacity knapsacks.
- A set of items $\left\{ \underbrace{\left(\frac{1}{2}, 5\right)}_{\times 2}, \underbrace{\left(\frac{1}{3}, 3\right)}_{\times 3}, \underbrace{(1, 18)}_{\times 1} \right\}$.

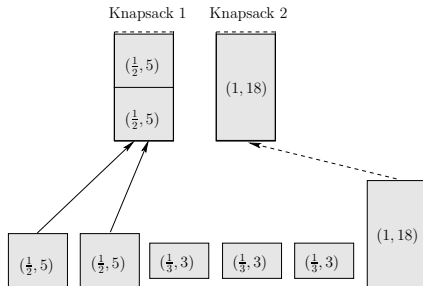


A possible solution: pack the “small” items.

Total value: 19

Example

- 2 unit-capacity knapsacks.
- A set of items $\left\{ \underbrace{\left(\frac{1}{2}, 5\right)}_{\times 2}, \underbrace{\left(\frac{1}{3}, 3\right)}_{\times 3}, \underbrace{(1, 18)}_{\times 1} \right\}$.



An optimal solution: pack the “big” items.

Total value: 28

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - **The setting: mechanism design**
 - The difficulty: unifying algorithms
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application
- 3 **Concluding Remarks**

Algorithmic mechanism design

- Studies algorithmic problems in scenarios where the input is presented by **strategic agents**.
- Focuses on the development of **truthful mechanisms**.

Algorithmic mechanism design

- Studies algorithmic problems in scenarios where the input is presented by **strategic agents**.
- Focuses on the development of **truthful mechanisms**.

Strategic agent

May declare any fallacious input in order to manipulate the algorithm in a way that will maximize its own utility.

Algorithmic mechanism design

- Studies algorithmic problems in scenarios where the input is presented by **strategic agents**.
- Focuses on the development of **truthful mechanisms**.

Strategic agent

May declare any fallacious input in order to manipulate the algorithm in a way that will maximize its own utility.

Truthful mechanism

A way to motivate the agents to truthfully report their inputs, e.g., compensating them by payments.

Strategic agent in MKP

Agents correspond to items. Agent i ...

- **May** be dishonest about its value, v_i .
- **Cannot** be untruthful about its size, s_j .

Strategic agent in MKP

Agents correspond to items. Agent i ...

- **May** be dishonest about its value, v_i .
- **Cannot** be untruthful about its size, s_i .

Truthful mechanism for MKP

Truthful mechanism \iff Monotone algorithm

Strategic agent in MKP

Agents correspond to items. Agent i ...

- **May** be dishonest about its value, v_i .
- **Cannot** be untruthful about its size, s_i .

Truthful mechanism for MKP

Truthful mechanism \iff Monotone algorithm

An algorithm \mathcal{A} is **monotone** if it satisfies the property that

- If \mathcal{A} packs item i having value of v_i
- Then \mathcal{A} would have packed i having value of $\tilde{v}_i \geq v_i$
- Assuming all the other values were fixed.

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - **The difficulty: unifying algorithms**
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application
- 3 **Concluding Remarks**

If all items had $s_i \in (0, \frac{1}{2}]$:

Algorithm SMALL

- Sort the items by decreasing **profit density ratios** (v_i/s_i).
- **Greedily** pack each item in some knapsack, until no feasible knapsack exists.

Algorithm SMALL is deterministic, 2-approx, and monotone.

If all items had $s_i \in (0, \frac{1}{2}]$:

Algorithm SMALL

- Sort the items by decreasing **profit density ratios** (v_i/s_i).
- **Greedy** pack each item in some knapsack, until no feasible knapsack exists.

Algorithm SMALL is deterministic, 2-approx, and monotone.

If all items had $s_i \in (\frac{1}{2}, 1]$:

Algorithm BIG

- Sort the items by decreasing **values** (v_i).
- **Greedy** pack each item in some knapsack, until no feasible knapsack exists.

Algorithm BIG is deterministic, 1-approx, and monotone.

If all items had $s_i \in (0, \frac{1}{2}]$:

Algorithm SMALL

- Sort the items by decreasing **profit density ratios** (v_i/s_i).
- **Greedy** pack each item in some knapsack, until no feasible knapsack exists.

Algorithm SMALL is deterministic, 2-approx, and monotone.

If all items had $s_i \in (\frac{1}{2}, 1]$:

Algorithm BIG

- Sort the items by decreasing **values** (v_i).
- **Greedy** pack each item in some knapsack, until no feasible knapsack exists.

Algorithm BIG is deterministic, 1-approx, and monotone.

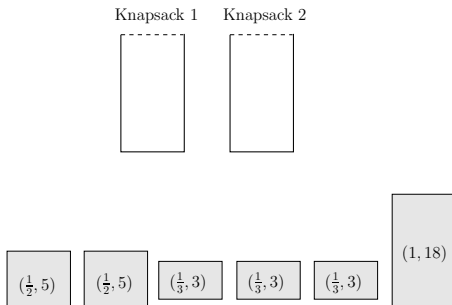
Given unrestrictive instance, what happens we execute each algorithm w.r.t. the relevant items and **pick the best solution?**

Algorithm $\text{MAX}\{\text{SMALL}, \text{BIG}\}$

- Deterministic, 3-approx, and...

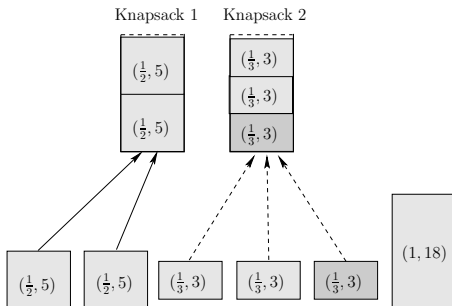
Algorithm MAX{SMALL, BIG}

- Deterministic, 3-approx, and...
- **Not monotone!**



Algorithm MAX{SMALL, BIG}

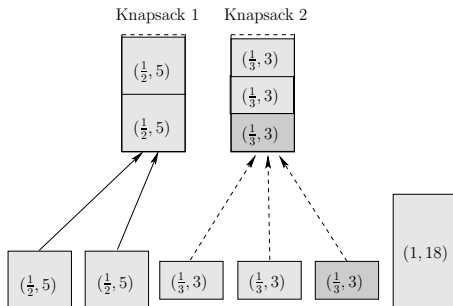
- Deterministic, 3-approx, and...
- **Not monotone!**



SMALL's overall value is 19, whereas **BIG's** overall value is 18.

Algorithm MAX{SMALL, BIG}

- Deterministic, 3-approx, and...
- **Not monotone!**

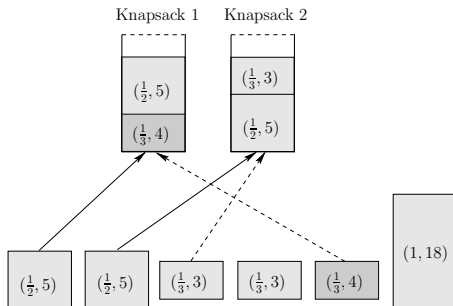


SMALL's overall value is 19, whereas **BIG's** overall value is 18.

Increasing the value of an item from 3 to 4...

Algorithm $\text{MAX}\{\text{SMALL}, \text{BIG}\}$

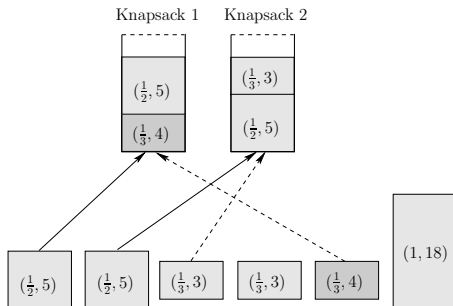
- Deterministic, 3-approx, and...
- **Not monotone!**



SMALL's overall value is 17, whereas **BIG**'s overall value is 18.

Algorithm MAX{SMALL, BIG}

- Deterministic, 3-approx, and...
- **Not monotone!**



SMALL's overall value is 17, whereas **BIG's** overall value is 18.

The mentioned item is **not selected**.

What happened?

Notice that monotonicity...

- guarantees that an agent **continues to be chosen** when it increases its value.
 - The item continues to be selected by SMALL.
- does not guarantee that the **outcome value increases** when the value of a selected agent increases.
 - The outcome value of SMALL decreased from 19 to 17.

What happened?

Notice that monotonicity...

- guarantees that an agent **continues to be chosen** when it increases its value.
 - The item continues to be selected by SMALL.
- does not guarantee that the **outcome value increases** when the value of a selected agent increases.
 - The outcome value of SMALL decreased from 19 to 17.

What can be done?

[MN '02] Work only with algorithms that are both monotone and **bitonic** (\approx having guarantees w.r.t. the outcome value).

What happened?

Notice that monotonicity...

- guarantees that an agent **continues to be chosen** when it increases its value.
 - The item continues to be selected by SMALL.
- does not guarantee that the **outcome value increases** when the value of a selected agent increases.
 - The outcome value of SMALL decreased from 19 to 17.

What can be done?

Very **restrictive** requirement... can we do anything else?

Outline

- 1 **Motivating Example**
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 **Our Results**
 - Unification framework for PIPs
 - Framework application
- 3 **Concluding Remarks**

The approach thus far

- 1 Partition a problem instance into sub-instances.
- 2 Execute a monotone algorithm on each sub-instance.
- 3 Pick the best sub-solution.

The approach thus far

- 1 Partition a problem instance into sub-instances.
- 2 Execute a monotone algorithm on each sub-instance.
- 3 Pick the best sub-solution.

The key idea - use two sets of algorithms:

- **Bitonic** alg's: used to decide which sub-instance to solve.
- **Monotone** alg's: used to solve the sub-instances.

The approach thus far

- 1 Partition a problem instance into sub-instances.
- 2 Execute a monotone algorithm on each sub-instance.
- 3 Pick the best sub-solution.

The key idea - use two sets of algorithms:

- **Bitonic** alg's: used to decide which sub-instance to solve.
 - They may not generate a feasible **integral** solution.
- **Monotone** alg's: used to solve the sub-instances.

The modified approach

- 1 Partition a problem instance into sub-instances.
- 2 Execute a **bitonic** algorithm on each sub-instance.
- 3 Execute a **monotone** algorithm on the sub-instance corresponding to the **best bitonic sub-solution**.

Now, the approximation guarantee also depends on the approximation properties of the bitonic algorithms.

The approach can be applied to packing integer programs

$$\begin{array}{ll}\mathbf{max} & v \cdot x \\ \mathbf{s.t.} & A \cdot x \leq b \\ & x \in \{0, 1\}^n\end{array}$$

- **Interpretation:** select max profit items respecting constraints.
- **MKP is a PIP:** constraints represent the knapsacks capacities.

The approach can be applied to packing integer programs

$$\begin{array}{ll} \mathbf{max} & v \cdot x \\ \mathbf{s.t.} & A \cdot x \leq b \\ & x \in \{0, 1\}^n \end{array}$$

- **Interpretation**: select max profit items respecting constraints.
- **MKP is a PIP**: constraints represent the knapsacks capacities.

Natural collection of bitonic algorithms

- Algorithms solving the LP-relaxation of the sub-instances.
- Approximation guarantee depends on the **integrality gaps** of the sub-instances.

Framework application for MKP

1 $\langle \text{sub}_{\text{SMALL}}, \text{sub}_{\text{BIG}} \rangle \leftarrow \text{items with } \langle s_i \in (0, \frac{1}{2}], s_i \in (\frac{1}{2}, 1] \rangle$

2

Framework application for MKP

- 1 $\langle \text{sub}_{\text{SMALL}}, \text{sub}_{\text{BIG}} \rangle \leftarrow \text{items with } \langle s_i \in (0, \frac{1}{2}], s_i \in (\frac{1}{2}, 1] \rangle$
- 2 solve the **LP-relaxations** corresponding to $\text{sub}_{\text{SMALL}}$ and sub_{BIG}
- 3 **if** $\text{LP}(\text{sub}_{\text{SMALL}}) \geq \text{LP}(\text{sub}_{\text{BIG}})$ **then**
- 4 |
- 5 **else**
- 6 |

Framework application for MKP

- 1 $\langle \text{sub}_{\text{SMALL}}, \text{sub}_{\text{BIG}} \rangle \leftarrow \text{items with } \langle s_i \in (0, \frac{1}{2}], s_i \in (\frac{1}{2}, 1] \rangle$
- 2 solve the **LP-relaxations** corresponding to $\text{sub}_{\text{SMALL}}$ and sub_{BIG}
- 3 **if** $\text{LP}(\text{sub}_{\text{SMALL}}) \geq \text{LP}(\text{sub}_{\text{BIG}})$ **then**
- 4 | return **SMALL**($\text{sub}_{\text{SMALL}}$)
- 5 **else**
- 6 | return **BIG**(sub_{BIG})

Framework application for MKP

- 1 $\langle \text{sub}_{\text{SMALL}}, \text{sub}_{\text{BIG}} \rangle \leftarrow \text{items with } \langle s_i \in (0, \frac{1}{2}], s_i \in (\frac{1}{2}, 1] \rangle$
- 2 solve the **LP-relaxations** corresponding to $\text{sub}_{\text{SMALL}}$ and sub_{BIG}
- 3 **if** $\text{LP}(\text{sub}_{\text{SMALL}}) \geq \text{LP}(\text{sub}_{\text{BIG}})$ **then**
- 4 | return **SMALL**($\text{sub}_{\text{SMALL}}$)
- 5 **else**
- 6 | return **BIG**(sub_{BIG})

Deterministic, constant approximation, and... **monotone**.

Outline

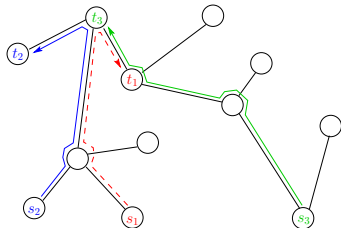
- 1 Motivating Example
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 Our Results
 - Unification framework for PIPs
 - **Framework application**
- 3 Concluding Remarks

Bandwidth allocation problem in tree networks

- The **input** consists of
 - A tree network.
 - A set of requests, each having a demand and a value.

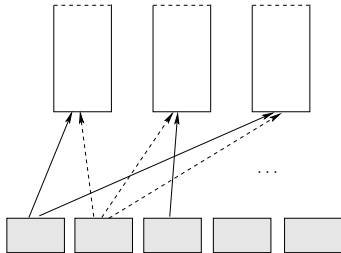
The **goal** is to select maximum value subset of requests, which can be simultaneously routed.

- We devise a monotone deterministic algorithm, achieving an approximation ratio of $O(\ln \ln m)$.



Multiple knapsack problem on bipartite graphs

- A generalization of MKP in which there are items-to-knapsacks **assignment restrictions**.
- We develop a monotone deterministic algorithm, achieving **constant** approximation guarantee.



Outline

- 1 Motivating Example
 - The problem: multiple knapsack
 - The setting: mechanism design
 - The difficulty: unifying algorithms
- 2 Our Results
 - Unification framework for PIPs
 - Framework application
- 3 Concluding Remarks

Open Questions

- **Improve the outcome** of the framework.

Open Questions

- **Improve the outcome** of the framework.
- Find **additional applications** of the framework.

Open Questions

- **Improve the outcome** of the framework.
- Find **additional applications** of the framework.
- **Improve the (truthful) approximation guarantees** for BAP and MKP.

Open Questions

- **Improve the outcome** of the framework.
- Find **additional applications** of the framework.
- **Improve the (truthful) approximation guarantees** for BAP and MKP.

Thank You

Slides will be available at my home page
<http://www.cs.tau.ac.il/~iftgam>