Implementation exercises for the course Heuristic Optimization

Alberto Franzin[1]
alberto.franzin@ulb.ac.be

IRIDIA, CoDE, ULB

March 14, 2018

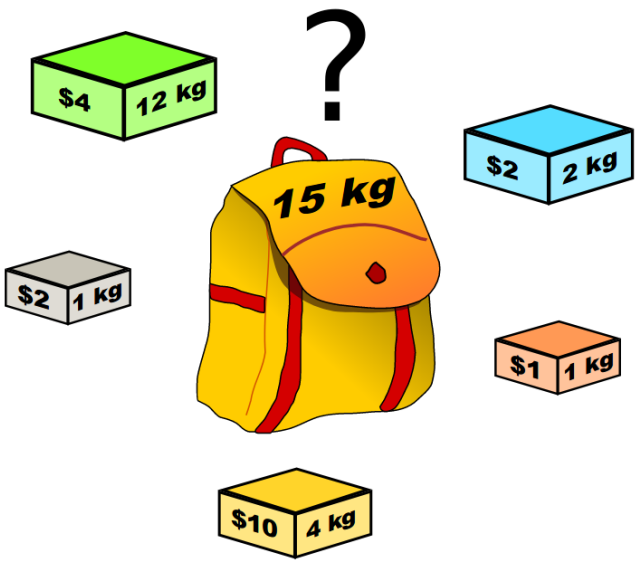
[1] Slides based on 2017 exercises by Federico Pagnozzi.

Exercise 1.1: Heuristics for the MKP

Implement constructive and perturbative local search algorithms for the MKP

1. Multidimensional 0-1 Knapsack Problem (MKP)
2. Constructive Heuristics
3. First-improvement and Best-improvement
4. Variable Neighbourhood Descent
5. Statistical Analysis

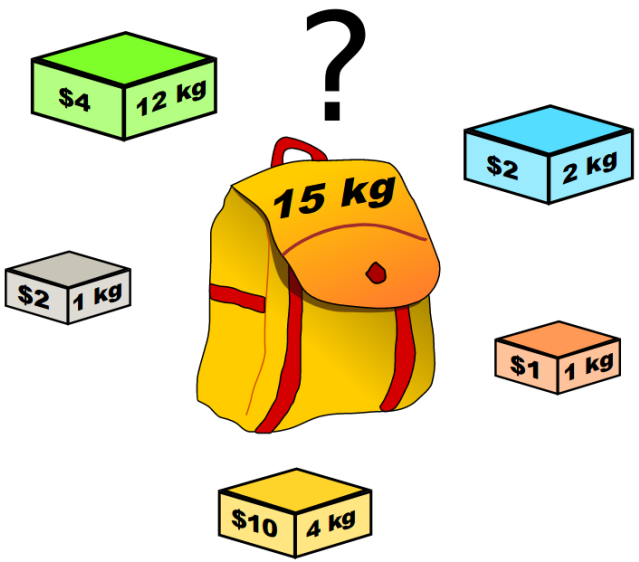
The 0-1 Knapsack Problem



Models many real-life allocation problems:

- a subset of item has to be chosen from a larger set
- each item has a *value*
- but items have also a *cost*.
- Resources are scarce: select the subset of items that give the highest cumulative value, while respecting the budget.

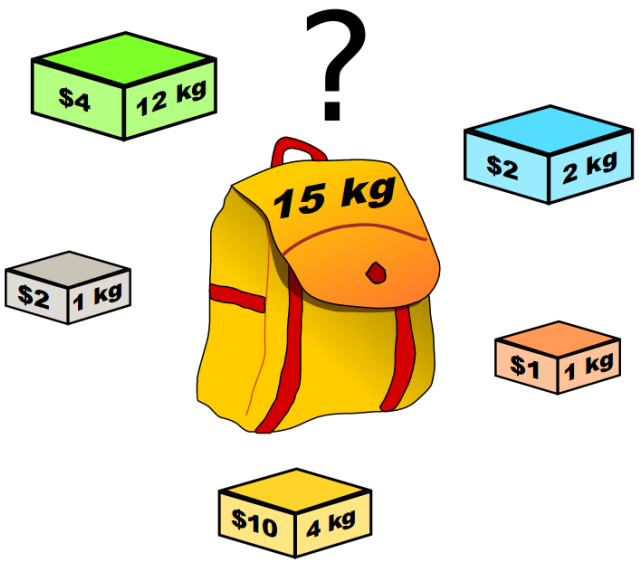
The 0-1 Knapsack Problem



Models many real-life allocation problems:

- a subset of item has to be chosen from a larger set
- each item has a *value*
- but items have also a *cost*.
- Resources are scarce: select the subset of items that give the highest cumulative value, while respecting the budget.

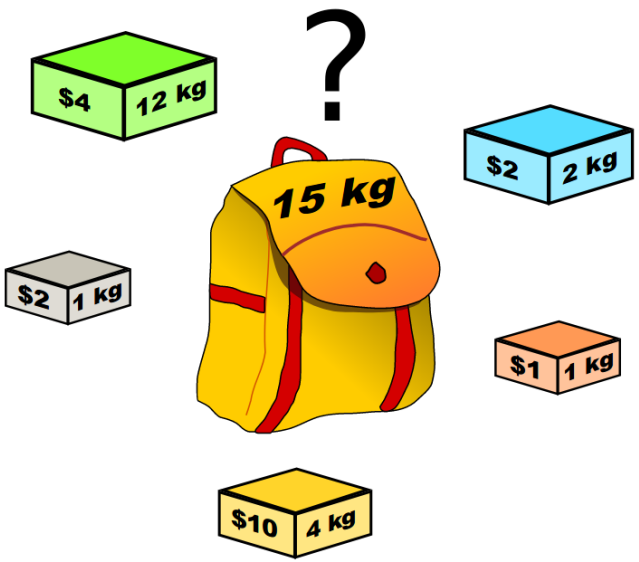
The 0-1 Knapsack Problem



Models many real-life allocation problems:

- a subset of item has to be chosen from a larger set
- each item has a *value*
- but items have also a *cost*.
- Resources are scarce: select the subset of items that give the highest cumulative value, while respecting the budget.

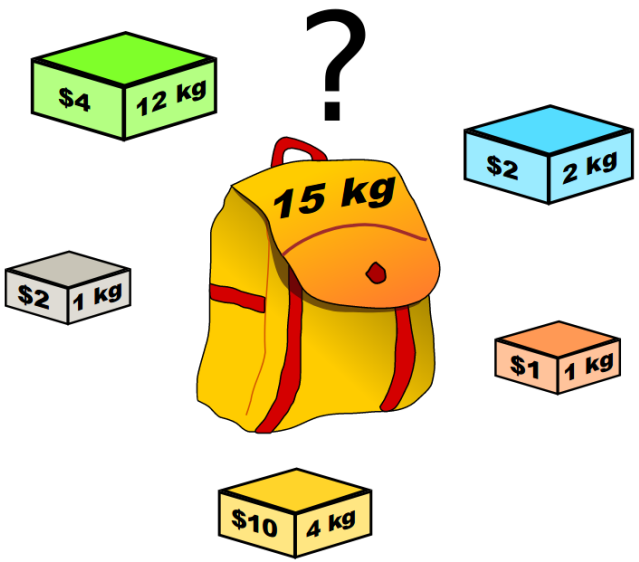
The 0-1 Knapsack Problem



Models many real-life allocation problems:

- a subset of item has to be chosen from a larger set
- each item has a *value*
- but items have also a *cost*.
- Resources are scarce: select the subset of items that give the highest cumulative value, while respecting the budget.

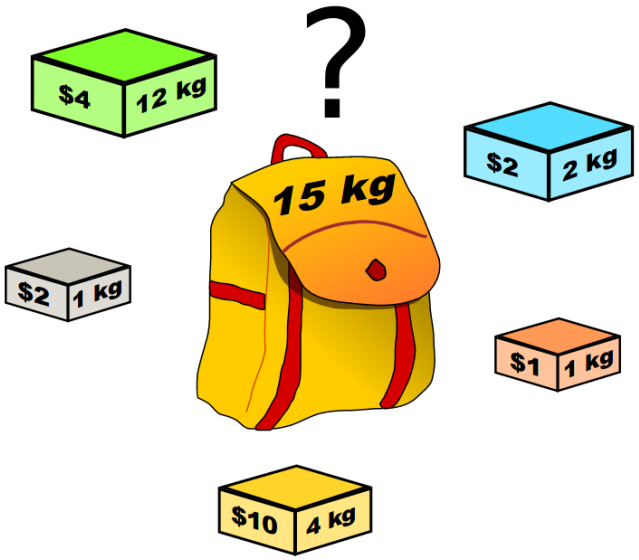
The 0-1 Knapsack Problem



Models many real-life allocation problems:

- a subset of item has to be chosen from a larger set
- each item has a *value*
- but items have also a *cost*.
- Resources are scarce: select the subset of items that give the highest cumulative value, while respecting the budget.

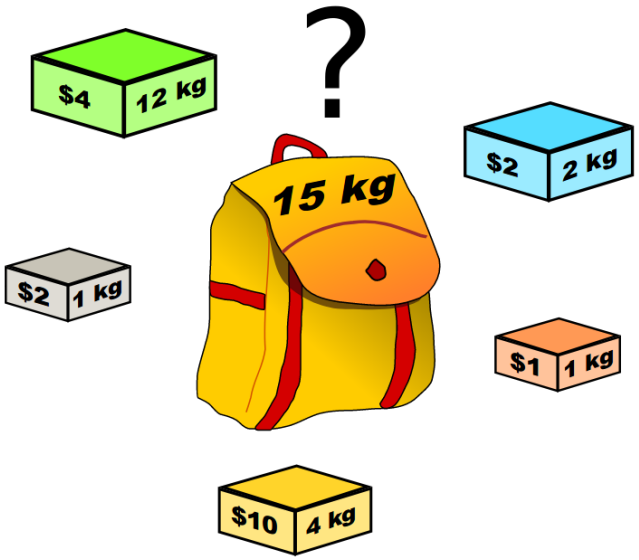
The Multidimensional 0-1 Knapsack Problem



Multidimensional: several resources involved

- each item is associated to a value and a set of resources needed: cost, weight, volume, . . .
- for every resource, the knapsack has a maximum capacity
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

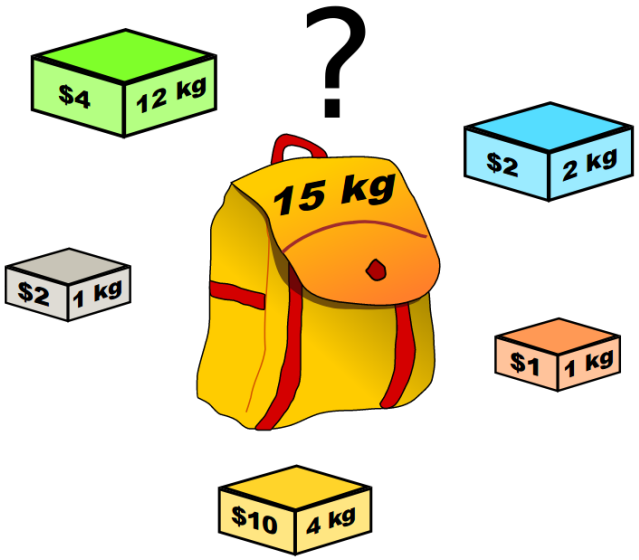
The Multidimensional 0-1 Knapsack Problem



Multidimensional: several resources involved

- each item is associated to a value and a set of resources needed: cost, weight, volume, . . .
- for every resource, the knapsack has a maximum capacity
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

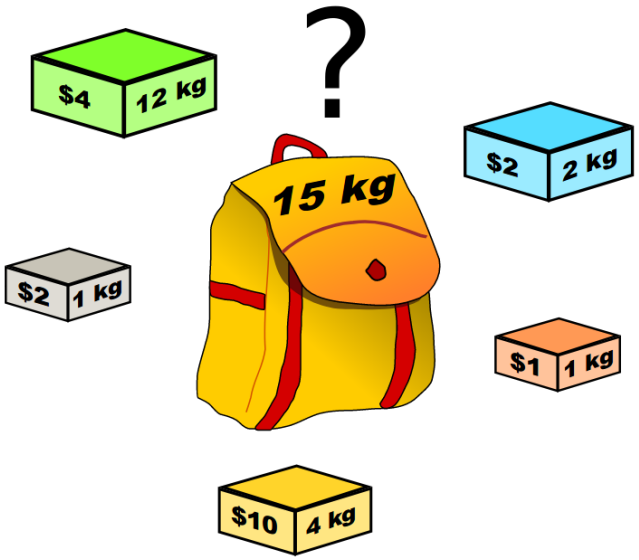
The Multidimensional 0-1 Knapsack Problem



Multidimensional: several resources involved

- each item is associated to a value and a set of resources needed: cost, weight, volume, . . .
- for every resource, the knapsack has a maximum capacity
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem



Multidimensional: several resources involved

- each item is associated to a value and a set of resources needed: cost, weight, volume, . . .
- for every resource, the knapsack has a maximum capacity
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

More precisely:

- we have *n items* and *m resources*
- we have a knapsack that can contain some (but not all) our items
  - the knapsack has a maximum *capacity* for every resource
- each item has associated *profit* or *value* p, and consumes a certain amount w_i for every resource i
- profits and weights are non-negative (so-called knapsack constraints)
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

More precisely:

- we have *n items* and *m resources*
- we have a knapsack that can contain some (but not all) our items
  - the knapsack has a maximum *capacity* for every resource
- each item has associated *profit* or *value* p, and consumes a certain amount w_i for every resource i
- profits and weights are non-negative (so-called knapsack constraints)
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

More precisely:

- we have *n items* and *m resources*
- we have a knapsack that can contain some (but not all) our items
  - the knapsack has a maximum *capacity* for every resource
- each item has associated *profit* or *value p*, and consumes a certain amount w_i for every resource i
- profits and weights are non-negative (so-called knapsack constraints)
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

More precisely:

- we have *n items* and *m resources*
- we have a knapsack that can contain some (but not all) our items
  - the knapsack has a maximum *capacity* for every resource
- each item has associated *profit* or *value p*, and consumes a certain amount w_i for every resource i
- profits and weights are non-negative (so-called knapsack constraints)
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

More precisely:

- we have *n items* and *m resources*
- we have a knapsack that can contain some (but not all) our items
  - the knapsack has a maximum *capacity* for every resource
- each item has associated *profit* or *value p*, and consumes a certain amount w_i for every resource i
- profits and weights are non-negative (so-called knapsack constraints)
- Goal: select the subset of items that give the highest profit, while respecting all the constraints.

The Multidimensional 0-1 Knapsack Problem

Given

A set of n items and m resources, where each item j has a profit $p_j \geq 0$ and consumes an amount $w_{ij} \geq 0$ of each resource i, and each resource i has a maximum capacity L_i that cannot be exceeded.

Objective

Find the subset of items that maximizes

$\sum_{j=1}^{n} p_j \cdot x_j$

subject to

$\sum_{j=1}^{n} w_{ij} \cdot x_j \leq L_i \; \forall i$

$x_j \in \{0, 1\}$.

The Multidimensional 0-1 Knapsack Problem

Given

A set of n items and m resources, where each item j has a profit $p_j \geq 0$ and consumes an amount $w_{ij} \geq 0$ of each resource i, and each resource i has a maximum capacity L_i that cannot be exceeded.

Objective

Find the subset of items that maximizes
$\sum_{j=1}^{n} p_j \cdot x_j$
subject to
$\sum_{j=1}^{n} w_{ij} \cdot x_j \leq L_i \; \forall i$
$x_j \in \{0, 1\}$.

Example

6 items, 10 constraints

Object profits: 100 600 1200 2400 500 2000

Constraints:

$$
\begin{array}{lcr}
8x_0 + 12x_1 + 13x_2 + 64x_3 + 22x_4 + 41x_5 & \leq & 80 \\
8x_0 + 12x_1 + 13x_2 + 75x_3 + 22x_4 + 41x_5 & \leq & 96 \\
3x_0 + 6x_1 + 4x_2 + 18x_3 + 6x_4 + 4x_5 & \leq & 20 \\
5x_0 + 10x_1 + 8x_2 + 32x_3 + 6x_4 + 12x_5 & \leq & 36 \\
5x_0 + 13x_1 + 8x_2 + 42x_3 + 6x_4 + 20x_5 & \leq & 44 \\
5x_0 + 13x_1 + 8x_2 + 48x_3 + 6x_4 + 20x_5 & \leq & 48 \\
0x_0 + 0x_1 + 0x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 10 \\
3x_0 + 0x_1 + 4x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 18 \\
3x_0 + 2x_1 + 4x_2 + 0x_3 + 8x_4 + 4x_5 & \leq & 22 \\
3x_0 + 2x_1 + 4x_2 + 8x_3 + 8x_4 + 4x_5 & \leq & 24
\end{array}
$$

Example

Solution: items $\{0, 2, 5\}$

Solution value: $100 + 1200 + 2000 = 3300$

No constraints violated: solution is feasible

$$
\begin{array}{lcr|r}
8x_0 + 12x_1 + 13x_2 + 64x_3 + 22x_4 + 41x_5 & \leq & 80 & (62) \\
8x_0 + 12x_1 + 13x_2 + 75x_3 + 22x_4 + 41x_5 & \leq & 96 & (62) \\
3x_0 + 6x_1 + 4x_2 + 18x_3 + 6x_4 + 4x_5 & \leq & 20 & (11) \\
5x_0 + 10x_1 + 8x_2 + 32x_3 + 6x_4 + 12x_5 & \leq & 36 & (25) \\
5x_0 + 13x_1 + 8x_2 + 42x_3 + 6x_4 + 20x_5 & \leq & 44 & (33) \\
5x_0 + 13x_1 + 8x_2 + 48x_3 + 6x_4 + 20x_5 & \leq & 48 & (33) \\
0x_0 + 0x_1 + 0x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 10 & (0) \\
3x_0 + 0x_1 + 4x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 18 & (7) \\
3x_0 + 2x_1 + 4x_2 + 0x_3 + 8x_4 + 4x_5 & \leq & 22 & (11) \\
3x_0 + 2x_1 + 4x_2 + 8x_3 + 8x_4 + 4x_5 & \leq & 24 & (11)
\end{array}
$$

Example

Solution: items $\{0, 3\}$

Solution value: $100 + 2400 = 2500$

Some constraints violated: solution is not feasible

$8x_0 + 12x_1 + 13x_2 + 64x_3 + 22x_4 + 41x_5$	$\leq$	80	(72)
$8x_0 + 12x_1 + 13x_2 + 75x_3 + 22x_4 + 41x_5$	$\leq$	96	(83)
$3x_0 + 6x_1 + 4x_2 + 18x_3 + 6x_4 + 4x_5$	$\leq$	20	(21)
$5x_0 + 10x_1 + 8x_2 + 32x_3 + 6x_4 + 12x_5$	$\leq$	36	(37)
$5x_0 + 13x_1 + 8x_2 + 42x_3 + 6x_4 + 20x_5$	$\leq$	44	(47)
$5x_0 + 13x_1 + 8x_2 + 48x_3 + 6x_4 + 20x_5$	$\leq$	48	(53)
$0x_0 + 0x_1 + 0x_2 + 0x_3 + 8x_4 + 0x_5$	$\leq$	10	(0)
$3x_0 + 0x_1 + 4x_2 + 0x_3 + 8x_4 + 0x_5$	$\leq$	18	(3)
$3x_0 + 2x_1 + 4x_2 + 0x_3 + 8x_4 + 4x_5$	$\leq$	22	(3)
$3x_0 + 2x_1 + 4x_2 + 8x_3 + 8x_4 + 4x_5$	$\leq$	24	(11)

Item 3 is problematic here: takes a lot of resources for some constraints

Example

Solution: items $\{0, 3\}$

Solution value: $100 + 2400 = 2500$

Some constraints violated: solution is not feasible

$8x_0 + 12x_1 + 13x_2 + 64x_3 + 22x_4 + 41x_5$	$\leq$	80	(72)
$8x_0 + 12x_1 + 13x_2 + 75x_3 + 22x_4 + 41x_5$	$\leq$	96	(83)
$3x_0 + 6x_1 + 4x_2 + 18x_3 + 6x_4 + 4x_5$	$\leq$	20	(21)
$5x_0 + 10x_1 + 8x_2 + 32x_3 + 6x_4 + 12x_5$	$\leq$	36	(37)
$5x_0 + 13x_1 + 8x_2 + 42x_3 + 6x_4 + 20x_5$	$\leq$	44	(47)
$5x_0 + 13x_1 + 8x_2 + 48x_3 + 6x_4 + 20x_5$	$\leq$	48	(53)
$0x_0 + 0x_1 + 0x_2 + 0x_3 + 8x_4 + 0x_5$	$\leq$	10	(0)
$3x_0 + 0x_1 + 4x_2 + 0x_3 + 8x_4 + 0x_5$	$\leq$	18	(3)
$3x_0 + 2x_1 + 4x_2 + 0x_3 + 8x_4 + 4x_5$	$\leq$	22	(3)
$3x_0 + 2x_1 + 4x_2 + 8x_3 + 8x_4 + 4x_5$	$\leq$	24	(11)

Item 3 is problematic here: takes a lot of resources for some constraints

Exercise 1.1: Heuristics for the MKP

Implement 3 constructive heuristics and 9 iterative improvements algorithms for the MKP

Exercise 1.1: Heuristics for the MKP

Implement 3 constructive heuristics and 9 iterative improvements algorithms for the MKP

- Constructive heuristics:
  1. Random
  2. Greedy
  3. Toyoda (profit/weight ratio)
- Iterative improvement with pivoting rules:
  1. First-improvement (FI)
  2. Best-improvement (BI)
- Variable Neighbourhood Descent:
  1. based either on FI or BI (your choice)

Exercise 1.1: Heuristics for the MKP

Implement 3 constructive heuristics and 9 iterative improvements algorithms for the MKP

- Constructive heuristics:
  1. Random
  2. Greedy
  3. Toyoda (profit/weight ratio)
- Iterative improvement with pivoting rules:
  1. First-improvement (FI)
  2. Best-improvement (BI)
- Variable Neighbourhood Descent:
  1. based either on FI or BI (your choice)

Don't implement 12 programs!

Reuse code and use command-line parameters

```
mkp --random --none
mkp --greedy --fi
...
```

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

- Random Insertion
- Greedy Insertion (deterministic)
- Insertion based on ratio profit/weight (Toyoda algorithm, deterministic)
- Will be used as initial solutions for FI, BI, VND
- All of them start from an empty solution, iteratively add items whilealways maintaining feasibility
- Alternative approach: start from / generate an infeasible solution, remove items until feasibility is reached (not considered in these exercises).

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Random Insertion

- Start from an empty solution
- Shuffle the items
- For each item in this “order”: add it if it does not violate any constraint

Greedy Insertion

- Start from an empty solution
- Sort the items by profit, highest profit first
- For each item in this order: add it if it does not violate any constraint

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- generic algorithm for binary mixed integer problems
- sorts the items according to their "convenience" (pseudo-utility)
- pseudo-utility defined as $\max_j p_j / V_j$, V_j aggregate consumption of resources
- several possible ways for defining V_j

For the 1-resource case ("classic" KP)

- simple: one weight per item $\Rightarrow p_j / w_j$
- gives a greedy approximation algorithm, $\rho = 2$
- not so clear how to extend it to the multidimensional case

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i$, $a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i, \; a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i, \; a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i$, $a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i$, $a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i$, $a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Constructive heuristics for the MKP

Toyoda algorithm

- consider $A = W/L$, that is, normalize in $[0, 1]$ each w_{ij} and store them in a new matrix A
- constraints now defined as: $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq 1 \; \forall i$, $a_{ij} = w_{ij}/L_i$
- define U as the vector of resources used so far
- $U_i = \sum_{j=1}^{n} a_{ij} \cdot x_j$
- $V_j = \sum_{i=1}^{m} a_{ij} \cdot U_i / ||U||$ (where $||U||$ is the euclidean norm of U).
- careful in the first iteration: $0/0 \Rightarrow$ consider only $V_j^0 = \sum_{i=1}^{m} a_{ij}$
- sort the non-selected items according to the pseudo-utility p_j/V_j
- scan the list of items, until you can add one item
- when you add one item, stop the cycle, update and repeat the process until you cannot add any new item

Toyoda algorithm: example (1/6)

Consider our example instance with 6 items, 10 constraints

Object profits: 100 600 1200 2400 500 2000

Constraints:

$$
\begin{array}{lcr}
8x_0 + 12x_1 + 13x_2 + 64x_3 + 22x_4 + 41x_5 & \leq & 80 \\
8x_0 + 12x_1 + 13x_2 + 75x_3 + 22x_4 + 41x_5 & \leq & 96 \\
3x_0 + 6x_1 + 4x_2 + 18x_3 + 6x_4 + 4x_5 & \leq & 20 \\
5x_0 + 10x_1 + 8x_2 + 32x_3 + 6x_4 + 12x_5 & \leq & 36 \\
5x_0 + 13x_1 + 8x_2 + 42x_3 + 6x_4 + 20x_5 & \leq & 44 \\
5x_0 + 13x_1 + 8x_2 + 48x_3 + 6x_4 + 20x_5 & \leq & 48 \\
0x_0 + 0x_1 + 0x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 10 \\
3x_0 + 0x_1 + 4x_2 + 0x_3 + 8x_4 + 0x_5 & \leq & 18 \\
3x_0 + 2x_1 + 4x_2 + 0x_3 + 8x_4 + 4x_5 & \leq & 22 \\
3x_0 + 2x_1 + 4x_2 + 8x_3 + 8x_4 + 4x_5 & \leq & 24
\end{array}
$$

Toyoda algorithm: example (2/6)

Normalize the constraints: $\sum_{i=1}^{m} a_{ij} \cdot x_j \leq 1 \; \forall j, \; a_{ij} = w_{ij}/L_i$

Object profits: 100 600 1200 2400 500 2000

Constraints:

$$
\begin{array}{lcl}
0.1x_0 + 0.15x_1 + 0.1625x_2 + 0.8x_3 + 0.275x_4 + 0.5125x_5 & \leq & 1 \\
0.0833x_0 + 0.125x_1 + 0.1354x_2 + 0.78125x_3 + 0.2292x_4 + 0.4271x_5 & \leq & 1 \\
0.15x_0 + 0.3x_1 + 0.2x_2 + 0.9x_3 + 0.3x_4 + 0.2x_5 & \leq & 1 \\
0.1389x_0 + 0.2778x_1 + 0.2222x_2 + 0.8889x_3 + 0.1667x_4 + 0.3333x_5 & \leq & 1 \\
0.1136x_0 + 0.2955x_1 + 0.1818x_2 + 0.9546x_3 + 0.1364x_4 + 0.4545x_5 & \leq & 1 \\
0.1041x_0 + 0.2708x_1 + 0.1667x_2 + 1x_3 + 0.125x_4 + 0.41667x_5 & \leq & 1 \\
0x_0 + 0x_1 + 0x_2 + 0x_3 + 0.8x_4 + 0x_5 & \leq & 1 \\
0.1667x_0 + 0x_1 + 0.2222x_2 + 0x_3 + 0.4444x_4 + 0x_5 & \leq & 1 \\
0.1364x_0 + 0.0909x_1 + 0.1818x_2 + 0x_3 + 0.3636x_4 + 0.1818x_5 & \leq & 1 \\
0.125x_0 + 0.0833x_1 + 0.1667x_2 + 0.3333x_3 + 0.3333x_4 + 0.1667x_5 & \leq & 1
\end{array}
$$

$U = [0, 0, \ldots, 0]$

Toyoda algorithm: example (3/6)

We start from an empty solution, $U = [0, 0, \ldots, 0]$, $||U|| = 0$, hence in the first iteration we compute only $V_j = \sum_{i=1}^{m} a_{ij}$ (that is, the sum of each column)

$V_0 = 0.1 + 0.0833 + 0.15 + 0.1389 + 0.1136 + 0.1041 + 0 + 0.1667 + 0.1364 + 0.125 = 1.118$

$V_1 = 1.593 \qquad V_2 = 1.639$

$V_3 = 5.658 \qquad V_4 = 3.1736 \qquad V_5 = 2.6926$

$p_0/V_0 = 100/1.118 = 89.44 \qquad p_1/V_1 = 376.575$

$p_2/V_2 = 732.006 \qquad p_3/V_3 = 424.1768$

$p_4/V_4 = 157.549 \qquad p_5/V_5 = 742.772$

Item 5 has the highest pseudo-utility, and adding it to the solution does not violate any constraint $\Rightarrow s = \{5\}$,

$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167] \leq [1, \ldots, 1]$

Toyoda algorithm: example (3/6)

We start from an empty solution, $U = [0, 0, \ldots, 0]$, $||U|| = 0$, hence in the first iteration we compute only $V_j = \sum_{i=1}^{m} a_{ij}$ (that is, the sum of each column)

$V_0 = 0.1 + 0.0833 + 0.15 + 0.1389 + 0.1136 + 0.1041 + 0 + 0.1667 + 0.1364 + 0.125 = 1.118$

$V_1 = 1.593 \qquad V_2 = 1.639$

$V_3 = 5.658 \qquad V_4 = 3.1736 \qquad V_5 = 2.6926$

$p_0/V_0 = 100/1.118 = 89.44 \qquad p_1/V_1 = 376.575$

$p_2/V_2 = 732.006 \qquad p_3/V_3 = 424.1768$

$p_4/V_4 = 157.549 \qquad p_5/V_5 = 742.772$

Item 5 has the highest pseudo-utility, and adding it to the solution does not violate any constraint $\Rightarrow s = \{5\}$,

$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167] \leq [1, \ldots, 1]$

Toyoda algorithm: example (3/6)

We start from an empty solution, $U = [0, 0, \ldots, 0]$, $||U|| = 0$, hence in the first iteration we compute only $V_j = \sum_{i=1}^{m} a_{ij}$ (that is, the sum of each column)

$V_0 = 0.1 + 0.0833 + 0.15 + 0.1389 + 0.1136 + 0.1041 + 0 + 0.1667 + 0.1364 + 0.125 = 1.118$

$V_1 = 1.593 \qquad V_2 = 1.639$

$V_3 = 5.658 \qquad V_4 = 3.1736 \qquad V_5 = 2.6926$

$p_0/V_0 = 100/1.118 = 89.44 \qquad p_1/V_1 = 376.575$

$p_2/V_2 = 732.006 \qquad p_3/V_3 = 424.1768$

$p_4/V_4 = 157.549 \qquad p_5/V_5 = 742.772$

Item 5 has the highest pseudo-utility, and adding it to the solution does not violate any constraint $\Rightarrow s = \{5\}$,

$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167] \leq [1, \ldots, 1]$

Toyoda algorithm: example (3/6)

We start from an empty solution, $U = [0, 0, \ldots, 0]$, $||U|| = 0$, hence in the first iteration we compute only $V_j = \sum_{i=1}^{m} a_{ij}$ (that is, the sum of each column)

$V_0 = 0.1 + 0.0833 + 0.15 + 0.1389 + 0.1136 + 0.1041 + 0 + 0.1667 + 0.1364 + 0.125 = 1.118$
$V_1 = 1.593 \qquad V_2 = 1.639$
$V_3 = 5.658 \qquad V_4 = 3.1736 \qquad V_5 = 2.6926$

$p_0/V_0 = 100/1.118 = 89.44 \qquad p_1/V_1 = 376.575$
$p_2/V_2 = 732.006 \qquad p_3/V_3 = 424.1768$
$p_4/V_4 = 157.549 \qquad p_5/V_5 = 742.772$
Item 5 has the highest pseudo-utility, and adding it to the solution does not violate any constraint $\Rightarrow s = \{5\}$,
$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167] \leq [1, \ldots, 1]$

Toyoda algorithm: example (4/6)

Second iteration: $s = \{5\}$, profit = 2000
$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167]$,
$||U|| = 1.018386$

From now on we have to consider U, $||U||$

$V_0 = (0.1*0.5125+0.0833*0.4271+\cdots+0.125*0.167)/1.018386 = 0.298$
$V_1 = 0.5503 \qquad V_2 = 0.4596$
$V_3 = 2.088 \qquad V_4 = 0.579$

$p_0/V_0 = 100/0.298 = 335.21$
$p_1/V_1 = 1090.379 \qquad p_2/V_2 = 2610.75$
$p_3/V_3 = 1149.67 \qquad p_4/V_4 = 862.93$

Item 2 has the highest pseudo-utility now, and adding it to the solution does not violate any constraint

Toyoda algorithm: example (4/6)

Second iteration: $s = \{5\}$, profit = 2000
$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167]$,
$||U|| = 1.018386$

From now on we have to consider U, $||U||$

$V_0 = (0.1*0.5125+0.0833*0.4271+\cdots+0.125*0.167)/1.018386 = 0.298$
$V_1 = 0.5503 \qquad V_2 = 0.4596$
$V_3 = 2.088 \qquad V_4 = 0.579$

$p_0/V_0 = 100/0.298 = 335.21$
$p_1/V_1 = 1090.379 \qquad p_2/V_2 = 2610.75$
$p_3/V_3 = 1149.67 \qquad p_4/V_4 = 862.93$

Item 2 has the highest pseudo-utility now, and adding it to the solution does not violate any constraint

Toyoda algorithm: example (4/6)

Second iteration: $s = \{5\}$, profit = 2000
$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167]$,
$||U|| = 1.018386$

From now on we have to consider U, $||U||$

$V_0 = (0.1*0.5125+0.0833*0.4271+\cdots+0.125*0.167)/1.018386 = 0.298$
$V_1 = 0.5503 \qquad V_2 = 0.4596$
$V_3 = 2.088 \qquad V_4 = 0.579$

$p_0/V_0 = 100/0.298 = 335.21$
$p_1/V_1 = 1090.379 \qquad p_2/V_2 = 2610.75$
$p_3/V_3 = 1149.67 \qquad p_4/V_4 = 862.93$

Item 2 has the highest pseudo-utility now, and adding it to the solution does not violate any constraint

Toyoda algorithm: example (4/6)

Second iteration: $s = \{5\}$, profit = 2000
$U = [0.5125, 0.4271, 0.2, 0.333, 0.4545, 0.4167, 0, 0, 0.1818, 0.167]$,
$||U|| = 1.018386$

From now on we have to consider U, $||U||$

$V_0 = (0.1*0.5125+0.0833*0.4271+\cdots+0.125*0.167)/1.018386 = 0.298$
$V_1 = 0.5503 \qquad V_2 = 0.4596$
$V_3 = 2.088 \qquad V_4 = 0.579$

$p_0/V_0 = 100/0.298 = 335.21$
$p_1/V_1 = 1090.379 \qquad p_2/V_2 = 2610.75$
$p_3/V_3 = 1149.67 \qquad p_4/V_4 = 862.93$

Item 2 has the highest pseudo-utility now, and adding it to the solution does not violate any constraint

Toyoda algorithm: example (5/6)

Third iteration: $s = \{2, 5\}$, profit = 3200
$U = [0.675, 0.5625, 0.4, 0.555, 0.636, 0.583, 0, 0.2222, 0.3636, 0.33]$,
$||U|| = 1.509452$

$V_0 = (0.1 * 0.6750 + 0.0833 * 0.5625 + \cdots + 0.125 * 0.3337)/1.509452 = 0.3398$
$V_1 = 0.5649 \qquad V_3 = 2.0769 \qquad V_4 = 0.6816$

$p_0/V_0 = 100/0.3398 = 294.3 \qquad p_1/V_1 = 1062.13$
$p_3/V_3 = 1155.549 \qquad p_4/V_4 = 733.54$

Item 3 has the highest pseudo-utility now, but adding it would violate some constraints!
$U = [1.475, 1.344, 1.3, 1.44, 1.59, 1.58, 0, 0.222, 0.3636, 0.667]$
Next one is item 1, that does not violate any constraint

Toyoda algorithm: example (5/6)

Third iteration: $s = \{2, 5\}$, profit = 3200
$U = [0.675, 0.5625, 0.4, 0.555, 0.636, 0.583, 0, 0.2222, 0.3636, 0.33]$,
$||U|| = 1.509452$

$V_0 = (0.1 * 0.6750 + 0.0833 * 0.5625 + \cdots + 0.125 * 0.3337)/1.509452 = 0.3398$
$V_1 = 0.5649 \qquad V_3 = 2.0769 \qquad V_4 = 0.6816$

$p_0/V_0 = 100/0.3398 = 294.3 \qquad p_1/V_1 = 1062.13$
$p_3/V_3 = 1155.549 \qquad p_4/V_4 = 733.54$

Item 3 has the highest pseudo-utility now, but adding it would violate some constraints!
$U = [1.475, 1.344, 1.3, 1.44, 1.59, 1.58, 0, 0.222, 0.3636, 0.667]$
Next one is item 1, that does not violate any constraint

Toyoda algorithm: example (5/6)

Third iteration: $s = \{2, 5\}$, profit = 3200
$U = [0.675, 0.5625, 0.4, 0.555, 0.636, 0.583, 0, 0.2222, 0.3636, 0.33]$,
$||U|| = 1.509452$

$V_0 = (0.1 * 0.6750 + 0.0833 * 0.5625 + \cdots + 0.125 * 0.3337)/1.509452 = 0.3398$
$V_1 = 0.5649 \qquad V_3 = 2.0769 \qquad V_4 = 0.6816$

$p_0/V_0 = 100/0.3398 = 294.3 \qquad p_1/V_1 = 1062.13$
$p_3/V_3 = 1155.549 \qquad p_4/V_4 = 733.54$

Item 3 has the highest pseudo-utility now, but adding it would violate some constraints!
$U = [1.475, 1.344, 1.3, 1.44, 1.59, 1.58, 0, 0.222, 0.3636, 0.667]$
Next one is item 1, that does not violate any constraint

Toyoda algorithm: example (5/6)

Third iteration: $s = \{2, 5\}$, profit = 3200
$U = [0.675, 0.5625, 0.4, 0.555, 0.636, 0.583, 0, 0.2222, 0.3636, 0.33]$,
$||U|| = 1.509452$

$V_0 = (0.1 * 0.6750 + 0.0833 * 0.5625 + \cdots + 0.125 * 0.3337)/1.509452 = 0.3398$
$V_1 = 0.5649 \qquad V_3 = 2.0769 \qquad V_4 = 0.6816$

$p_0/V_0 = 100/0.3398 = 294.3 \qquad p_1/V_1 = 1062.13$
$p_3/V_3 = 1155.549 \qquad p_4/V_4 = 733.54$

Item 3 has the highest pseudo-utility now, but adding it would violate some constraints!
$U = [1.475, 1.344, 1.3, 1.44, 1.59, 1.58, 0, 0.222, 0.3636, 0.667]$
Next one is item 1, that does not violate any constraint

Toyoda algorithm: example (5/6)

Third iteration: $s = \{2, 5\}$, profit = 3200
$U = [0.675, 0.5625, 0.4, 0.555, 0.636, 0.583, 0, 0.2222, 0.3636, 0.33]$,
$||U|| = 1.509452$

$V_0 = (0.1 * 0.6750 + 0.0833 * 0.5625 + \cdots + 0.125 * 0.3337)/1.509452 = 0.3398$
$V_1 = 0.5649 \qquad V_3 = 2.0769 \qquad V_4 = 0.6816$

$p_0/V_0 = 100/0.3398 = 294.3 \qquad p_1/V_1 = 1062.13$
$p_3/V_3 = 1155.549 \qquad p_4/V_4 = 733.54$

Item 3 has the highest pseudo-utility now, but adding it would violate some constraints!
$U = [1.475, 1.344, 1.3, 1.44, 1.59, 1.58, 0, 0.222, 0.3636, 0.667]$
Next one is item 1, that does not violate any constraint

Toyoda algorithm: example (6/6)

Fourth iteration: $s = \{1, 2, 5\}$, profit = 3800
$U = [0.825, 0.6875, 0.7, 0.833, 0.932, 0.854, 0, 0.222, 0.4545, 0.417]$,
$||U|| = 2.089237$

$V_0 = (0.1 * 0.825 + 0.0833 * 0.6875 + \cdots + 0.125 * 0.417)/2.089237 = 0.338$
$V_3 = 2.13 \qquad V_4 = 0.656$

$p_0/V_0 = 100/0.338 = 295.72 \qquad p_3/V_3 = 1126.7 \qquad p_4/V_4 = 762.5$

In order, we try to add items 3, 4, 0, but we cannot maintain feasibility. We're done.

Our final solution is $s = \{1, 2, 5\}$, with a profit of 3800.

Toyoda algorithm: example (6/6)

Fourth iteration: $s = \{1, 2, 5\}$, profit = 3800
$U = [0.825, 0.6875, 0.7, 0.833, 0.932, 0.854, 0, 0.222, 0.4545, 0.417]$,
$||U|| = 2.089237$

$V_0 = (0.1 * 0.825 + 0.0833 * 0.6875 + \cdots + 0.125 * 0.417)/2.089237 = 0.338$
$V_3 = 2.13 \qquad V_4 = 0.656$

$p_0/V_0 = 100/0.338 = 295.72 \qquad p_3/V_3 = 1126.7 \qquad p_4/V_4 = 762.5$

In order, we try to add items 3, 4, 0, but we cannot maintain feasibility. We're done.

Our final solution is $s = \{1, 2, 5\}$, with a profit of 3800.

Toyoda algorithm: example (6/6)

Fourth iteration: $s = \{1, 2, 5\}$, profit = 3800
$U = [0.825, 0.6875, 0.7, 0.833, 0.932, 0.854, 0, 0.222, 0.4545, 0.417]$,
$||U|| = 2.089237$

$V_0 = (0.1 * 0.825 + 0.0833 * 0.6875 + \cdots + 0.125 * 0.417)/2.089237 = 0.338$
$V_3 = 2.13 \qquad V_4 = 0.656$

$p_0/V_0 = 100/0.338 = 295.72 \qquad p_3/V_3 = 1126.7 \qquad p_4/V_4 = 762.5$

In order, we try to add items 3, 4, 0, but we cannot maintain feasibility. We're done.

Our final solution is $s = \{1, 2, 5\}$, with a profit of 3800.

Toyoda algorithm: example (6/6)

Fourth iteration: $s = \{1, 2, 5\}$, profit = 3800
$U = [0.825, 0.6875, 0.7, 0.833, 0.932, 0.854, 0, 0.222, 0.4545, 0.417]$,
$||U|| = 2.089237$

$V_0 = (0.1 * 0.825 + 0.0833 * 0.6875 + \cdots + 0.125 * 0.417)/2.089237 = 0.338$
$V_3 = 2.13 \qquad V_4 = 0.656$

$p_0/V_0 = 100/0.338 = 295.72 \qquad p_3/V_3 = 1126.7 \qquad p_4/V_4 = 762.5$

In order, we try to add items 3, 4, 0, but we cannot maintain feasibility. We're done.

Our final solution is $s = \{1, 2, 5\}$, with a profit of 3800.

Toyoda algorithm: example (6/6)

Fourth iteration: $s = \{1, 2, 5\}$, profit = 3800
$U = [0.825, 0.6875, 0.7, 0.833, 0.932, 0.854, 0, 0.222, 0.4545, 0.417]$,
$||U|| = 2.089237$

$V_0 = (0.1 * 0.825 + 0.0833 * 0.6875 + \cdots + 0.125 * 0.417)/2.089237 = 0.338$
$V_3 = 2.13 \qquad V_4 = 0.656$

$p_0/V_0 = 100/0.338 = 295.72 \qquad p_3/V_3 = 1126.7 \qquad p_4/V_4 = 762.5$

In order, we try to add items 3, 4, 0, but we cannot maintain feasibility. We're done.

Our final solution is $s = \{1, 2, 5\}$, with a profit of 3800.

Exercise 1.1: Heuristics for the MKP

Iterative Improvement

$s :=$ `GenerateInitialSolution()`

while s is not a local optimum **do**

  choose a neighbour $s' \in \mathcal{N}(s)$ such that $F(s') \geq F(s)$

  $s := s'$

Exercise 1.1: Heuristics for the MKP

Iterative Improvement

$s :=$ `GenerateInitialSolution()`

while s is not a local optimum **do**

  choose a neighbour $s' \in \mathcal{N}(s)$ such that $F(s') \geq F(s)$

  $s := s'$

Which neighbour to choose? Pivoting rule

- **Best Improvement:** choose best from all neighbours of s
  - ✔ Better quality
  - ✘ Requires evaluation of all neighbours in each step
- **First improvement:** evaluate neighbours in fixed order and choose first improving neighbour.
  - ✔ More efficient
  - ✘ Order of evaluation may impact quality / performance

Exercise 1.1: Heuristics for the MKP

Iterative Improvement

$s :=$ `GenerateInitialSolution()`

while s is not a local optimum **do**

  choose a neighbour $s' \in \mathcal{N}(s)$ such that $F(s') \geq F(s)$

  $s := s'$

Which neighbour to choose? Pivoting rule

- **Best Improvement:** choose best from all neighbours of s
  - ✔ Better quality
  - ✘ Requires evaluation of all neighbours in each step
- **First improvement:** evaluate neighbours in fixed order and choose first improving neighbour.
  - ✔ More efficient
  - ✘ Order of evaluation may impact quality / performance

Exercise 1.1: Heuristics for the MKP

Iterative Improvement

$s :=$ `GenerateInitialSolution()`

while s is not a local optimum **do**

  choose a neighbour $s' \in \mathcal{N}(s)$ such that $F(s') \geq F(s)$

  $s := s'$

Which neighbour to choose? Pivoting rule

- **Best Improvement:** choose best from all neighbours of s
  - ✔ Better quality
  - ✘ Requires evaluation of all neighbours in each step
- **First improvement:** evaluate neighbours in fixed order and choose first improving neighbour.
  - ✔ More efficient
  - ✘ Order of evaluation may impact quality / performance

Exercise 1.1: Heuristics for the MKP

Variable Neighbourhood Descent

- Based on either FI or BI (your choice)
- Evaluates increasingly large neighbourhoods

Initial solution

- Random constructive heuristic
- Greedy constructive heuristic
- Toyoda algorithm

Exercise 1.1: Heuristics for the MKP

Variable Neighbourhood Descent

- Based on either FI or BI (your choice)
- Evaluates increasingly large neighbourhoods

Initial solution

- Random constructive heuristic
- Greedy constructive heuristic
- Toyoda algorithm

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

Many neighbourhoods are possible for the MKP:

- remove k items and add k different items
- remove k items and add as many different items as possible
- $\rightarrow$ in which order?
- add k items, make the solution non feasible, then remove k different items
- add k items, make the solution non feasible, then remove as many different items as needed
- $\rightarrow$ in which order?

For this project, the neighbourhood is given by the set of moves that remove k items and add as many different items as possible, according to the order specified as follows.

Every solution accepted has to be feasible.

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP

- Remove k items
  - For FI, BI, $k = 1$
  - For VND, $k = 1, 2, 3$ (all the combination of k items are possible!)
- BUT: order has to be preserved in the same neighbourhood evaluation, hence:
- Shuffle the list of items in the solution
- Remove the items in this order
- FI: accept a move as soon as you find an improving one
- BI: scan the whole neighbourhood, record the best move, implement the best move at the end of the neighborhood scan
- Order for the reinsertion: according to the initial solution used
  - for Random: shuffle the list of non-inserted items
  - for greedy: highest profit first
  - for Toyoda: according to pseudo-utility
- use delta evaluation for both obj. function and constraint checking

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is [3]
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is [3]
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is $[3]$
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is $[3]$
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is $[3]$
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is $[3]$
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (1/2)

Consider our example instance, and an initial solution generated using the greedy criterion, and a BI rule: $s = \{3\}$, profit = 2400.

First iteration:

- shuffle the list of items in s
- shuffled list is $[3]$
- sort list of non-inserted items by profit: $[5, 2, 1, 4, 0]$
- remove first item from shuffled s: $\{\}$
- try to add the non-selected items, one by one, ordered by profit
- we can add 5, 2, 1, but not 4 or 0 (check feasibility). Hence: $s' = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$
- we cannot remove any more items from the shuffled s, so we have evaluated the whole neighbourhood. s' is feasible and it improves over s, so we accept it.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

Neighbourhood for the MKP: example (2/2)

$s = \{1, 2, 5\}$, profit = 3800, not inserted $\{0, 3, 4\}$

Next iteration, shuffle s: $[2, 5, 1]$. The other items sorted by profit are $[3, 4, 0]$. Hence:

- Remove item 2 from s, $'s = \{1, 5\}$, try to add 3, then 4, then 0
- Remove item 5 from s, $'s = \{1, 2\}$, try to add 3, then 4, then 0
- Remove item 1 from s, $'s = \{2, 5\}$, try to add 3, then 4, then 0

Neighbourhood scan completed, no improving move found: stop.

To recap:

Implement 3 constructive heuristics:

- Random
- Greedy
- Toyoda algorithm

Implement 3 FI heuristics:

- Random init. sol. + FI with random insertion
- Greedy init. sol. + FI with greedy insertion
- Toyoda init. sol. + FI with Toyoda insertion

Implement 3 BI heuristics:

- Random init. sol. + BI with random insertion
- Greedy init. sol. + BI with greedy insertion
- Toyoda init. sol. + BI with Toyoda insertion

Implement 3 VND heuristics (based either on FI or BI):

- Random init. sol. + VND with random insertion
- Greedy init. sol. + VND with greedy insertion
- Toyoda init. sol. + VND with Toyoda insertion

Material

Material provided on the course homepage:

- C code for
  - reading an MKP instance
  - computing the objective function value
- you can use the code as is, incapsulate it into C++ classes, rewrite it in Java
- 60 problem instances
  - + example instance for debugging
- list of best known values
- exercise sheet and this slide deck

Instance format

http://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html

- in the first line: problem size
  - number of items n
  - number of constraints m
  - value of the optimal solution (0 if not known)
- the list of n profits p_j (on multiple lines)
- the list of $n \times m$ weights w_{ij} (row-wise, on multiple lines)
- the list of m capacities L_i (on multiple lines)

Exercise 1.1: Heuristics for the MKP

Instances

- Run the Random constructive heuristic and the II based on it 15 times for each instance, and average the results
- Run the Greedy and Toyoda constructive heuristics and the II based on them once per each instance

Experiments

For each algorithm k and instance i compute:

1. Relative percentage deviation $\Delta_{ki} = 100 \cdot \frac{\text{profit}_{ki} - \text{best-known}_i}{\text{best-known}_i}$
2. Computation time (t_{ki})
3. (using the average values when the Random constructive heuristic is involved)

Exercise 1.1: Heuristics for the MKP

Instances

- Run the Random constructive heuristic and the II based on it 15 times for each instance, and average the results
- Run the Greedy and Toyoda constructive heuristics and the II based on them once per each instance

Experiments

For each algorithm k and instance i compute:

1. Relative percentage deviation $\Delta_{ki} = 100 \cdot \dfrac{\text{profit}_{ki} - \text{best-known}_i}{\text{best-known}_i}$
2. Computation time (t_{ki})
3. (using the average values when the Random constructive heuristic is involved)

Exercise 1.1: Heuristics for the MKP

Experiments

For each algorithm k and instance i compute:

1. Relative percentage deviation $\Delta_{ki} = 100 \cdot \frac{\text{profit}_{ki} - \text{best-known}_i}{\text{best-known}_i}$
2. Computation time (t_{ki})
3. (using the average values when the Random constructive heuristic is involved)

Report for each algorithm k

- Average relative percentage deviation
- Sum of computation time across all instances of a same size

Exercise 1.1: Heuristics for the MKP

Experiments

For each algorithm k and instance i compute:

1. Relative percentage deviation $\Delta_{ki} = 100 \cdot \dfrac{\text{profit}_{ki} - \text{best-known}_i}{\text{best-known}_i}$
2. Computation time (t_{ki})
3. (using the average values when the Random constructive heuristic is involved)

Report for each algorithm k

- Average relative percentage deviation
- Sum of computation time across all instances of a same size

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Statistical test

- Paired t-test
- Wilcoxon signed-rank test
- compare the results across the same size

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.
- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.
  Example: For the Wilcoxon signed-rank test, the null hypothesis is that 'the median of the differences is zero'.
- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.
  Typical values of α are 0.05 or 0.01.

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.
- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.
  Example: For the Wilcoxon signed-rank test, the null hypothesis is that 'the median of the differences is zero'.
- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.
  Typical values of α are 0.05 or 0.01.

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.
- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.
  Example: For the Wilcoxon signed-rank test, the null hypothesis is that 'the median of the differences is zero'.
- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.
  Typical values of α are 0.05 or 0.01.

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.
- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.
  Example: For the Wilcoxon signed-rank test, the null hypothesis is that 'the median of the differences is zero'.
- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.
  Typical values of α are 0.05 or 0.01.

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.
- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.
  Example: For the Wilcoxon signed-rank test, the null hypothesis is that 'the median of the differences is zero'.
- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.
  Typical values of α are 0.05 or 0.01.

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (2)

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.
- The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level.
- Most common statistical hypothesis tests are already implemented in statistical software such as the *R software environment* (http://www.r-project.org/).

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (2)

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.
- The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level.
- Most common statistical hypothesis tests are already implemented in statistical software such as the *R software environment* (http://www.r-project.org/).

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Background: Statistical hypothesis tests (2)

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.
- The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level.
- Most common statistical hypothesis tests are already implemented in statistical software such as the *R software environment* (http://www.r-project.org/).

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

Exercise 1.1: Heuristics for the MKP

Is there a statistically significant difference between the solution quality generated by the different algorithms?

Example in R

```
best.known <- read.csv ("bestSolutions.txt")
a.cost <- read.table("ii-best-ex-rand.dat")$V1
a.cost <- 100 * (a.cost - best.known) / best.known$BS
b.cost <- read.table("ii-best-ins-rand.dat")$V1
b.cost <- 100 * (b.cost - best.known) / best.known$BS
t.test (a.cost, b.cost, paired=T)$p.value
[1] 0.8819112
wilcox.test (a.cost, b.cost, paired=T)$p.value
[1] 0.0019212
```

- Instances and barebone code available at:
  http://iridia.ulb.ac.be/~stuetzle/Teaching/HO/
- Deadline is April 15 (23:59)
- Questions in the meantime?
  `alberto.franzin@ulb.ac.be`