# Project - Multidimensional Unitary-profit Precedence-constrained Knapsack Problem

Lucas Guesser Targino da Silva - RA: 203534

July 16, 2022

**Abstract**

We present a generalization of the classic 0-1 Knapsack Problem. In this problem, the weight of the items are multidimensional vectors and so is the knapsack capacity, the profit of all items is equal to one, and the items are required to be added in a certain order. That problem is refered as *Multidimensional Unitary-profit Precedence-constrained Knapsack Problem (MEPKP)*. Three solution approaches are proposed: an Integer Linear Programming for a exact solution; a greedy algorithm for a fast solution; a Greedy Randomized Adaptive Search Procedures (GRASP) and Tabu Search (TS) for a near optimal solution. The three approaches will be compared in terms of quality of the solution and computational time with randomly generated instances.

# 1 Problem Statement

## 1.1 Input

1. a directed acyclic graph $G = \langle V, E \rangle$;
2. a multi-dimensional weight function $w : V \to \mathbb{Z}_{>}^{n_w}$, where $n_w \in \mathbb{N}$;

   We will usually write $w_v = w(v)$
3. a maximum capacity of the knapsack $W \in \mathbb{Z}_{>}^{n_w}$;

Besides that, one requires the input to satisfy the constraints below [1], otherwise the problem would be trivial:

1. $w_v \leqslant W$: the weight of each vertex must be smaller than the knapsack capacity;
2. $\sum_{v \in V} w_v \geqslant W$: the weight of all vertices combined must be greater than the knapsack capacity;

### 1.1.1 Partial Order

**Definition 1** (Partial Order on Directed Acyclics Graph)**.** Given a directed acyclic graph $G = \langle V, E \rangle$, we define the set:

$$\prec = \{\langle v, v' \rangle : \text{there is a path from the second to the first}\} \tag{1}$$

and so $\prec$ is a partial order over the set $V$.

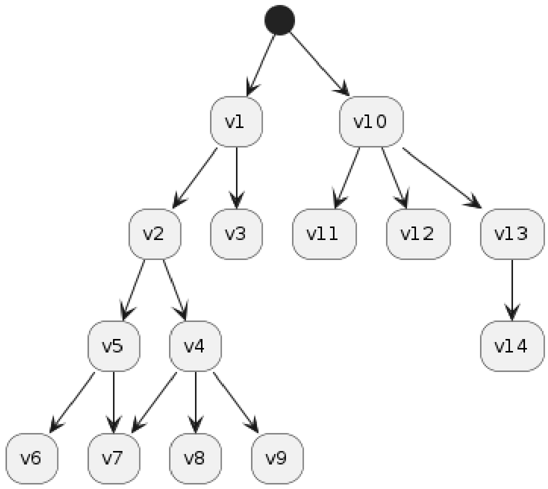


Figure 1: Example of a directed acyclic graph. The black dot indicates the root vertices. For this case, the induced partial order satisfy: $v5 \prec v2$, $v7 \prec v1$, $v14 \prec v10$.

### 1.2 Output

A subset $S \subseteq V$ of the vertices which satisfy:

$$\sum_{v \in S} w_v \leqslant W \tag{2}$$

$$\forall v \left( v \in S \rightarrow \forall v' \left( v \prec v' \rightarrow v' \in S \right) \right) \tag{3}$$

Equation (2) states the total weight of all vertices in the solution set $S$ must not be greater than the weight limit $W$. It is called Capacity-constraint.

Equation (3)[1] says that, if a $v$ is included in the solution, then all the $v'$ greater than it (in the sense of the partial order $\prec$) must also be included. It is called Precedence-constraint.

### 1.3 Objective

Find $S$ that maximizes $|S|$. In other words: find the solution with the maximum number of vertices.

## 2 State of the Art

In [3], the authors proposes a memory based GRASP for 0-1 quadratic knapsack problem with restart and a simple tabu search algorithm is proposed to overcome the limitations of local optimality in order to find near optimal solutions. In that paper, numerical tests on benchmark instances demonstrate the effectiveness and efficiency of the proposed methodology which outperform the Mini-Swarm heuristic in terms of the success ratio, relative percentage deviation and computational time.

[1] It is a First-order logic expression [2].

In [4], the authors reported the implementation of an efficient TS method based on the oscillation strategy and definition of a promising zone, a zone which englobes all feasible solutions plus all unfeasible solutions bordering the unfeasible solutions, for solving the O-1 MKP which has been tested on standard test problems from [5, 6] and [7]. Optimal solutions were successfully obtained for all instances and the previously best known solutions were improved for five of the last seven instances. These numerical results were claimed to confirm the merit of tabu tunneling approaches to generate solutions of high quality for 0-1 multiknapsack problems. Moreover, these results (like those of [7]) are claimed to establish that the oscillation strategy is efficient to balance the interaction between intensification and diversification strategies of TS.

In [1], the authors used a lagrangean relaxation on the precedence-constrained and the sub-gradient method to solve the problem faster then use a "pegging" test to guarantee optimality.

### 2.1 0-1 Knapsack Problem

We present it because of its simplicity, relavance in the literature, and similarity with the problem proposed in this problem. In [8], the authors define the 0-1 Knapsack Problem (0-1KP):

$$
\begin{aligned}
\max \qquad & \sum_{j=1}^{n} p_j x_j && (4)\\
\text{subjected to} \qquad & \sum_{j=1}^{n} w_j x_j \leq c \\
x_j \in \{0,1\} \quad & \forall j \in \{1, \ldots, n\}
\end{aligned}
$$

in which $p_j$ and $w_j$ are known as the profit and the weight of the item $j$, respectively.

The problem proposed here is a knapsack problem adapted to satisfy two extra constraints: precedence-constrained and multi-dimensional weights. Besides, its profits are all one.

## 3 Solving Methodologies

In this section, one presents all the methodologies proposed to solve the MUPKP.

### 3.1 Integer Linear Programming Model

#### 3.1.1 Decision Variables

For each vertex $v \in V$ of the input graph $G = \langle V, E \rangle$, we define the binary variable $x_v \in \{0, 1\}$ which indicates whether $v$ is in the solution $S$:

$$x_v = \begin{cases} 1 & , v \in S \\ 0 & , v \notin S \end{cases} \tag{5}$$

#### 3.1.2 Mathematical Model

Below, Equation (6) is the objective function: maximize the number of vertices in the solution. Equation (7) is the Capacity-Constraint of Equation (2). Equation (8) is the Precedence-Constraint of Equation (3): if a vertex $v$ is in the solution, then all vertices $v'$ "above" it must also be in the solution.

$$\max_{S \subseteq V} \sum_{v \in S} x_v \tag{6}$$

$$s.t. \sum_{v \in S} x_v w_v \leqslant W \tag{7}$$

$$x_v \leqslant x_{v'} \quad \forall v \prec v' \tag{8}$$

$$x \in \{0, 1\}^{n_w} \tag{9}$$

## 3.2 Greedy Criteria

Since the problem is a unitary-profit, the objective function provides no information about which vertex is better to add to the solution. Being more practical, for designing an algorithm, making decisions based solely on the objective function is useless and meaningless. For that reason, we propose the following auxiliary function $g$, called Greedy Criteria:

**Definition 2** (Greedy Criteria)**.** Let be a vertex and $w_v$ its weight. The Greedy Criteria is the function $g : V \to \mathbb{Z}_{>}$ which associates a vertex $v \in V$ to the entry of its weight vector $w_v \in \mathbb{Z}_{>}^{n_w}$ with the highest value:

$$g(v) = \max(w_v) \tag{10}$$

In metaheuristics, we are usually interested in greedy strategies. For the approaches analyzed in this project, the greedy criteria is going to be: select the vertex which minimizes the value of $g$. The intuition behind such choice is clear: we want to add vertices which weight as little as possible.

Notice that the above definition is not the only one available. One could choose to use the norm 2 or average value of the weight vector $w_v$. The election of the maximum is relies on the intuition that "averages" might fill too much one of the dimensions of the weight while leaving others free. That would cause early stop of the algorithms. The maximum function, on the other hand, ensures that dominating values of the dimension of the weight are properly handled.

Of course, using the maximum function has drawbacks as well. Since it looks only to the most loaded entry, between two vertices with the same value of the greedy function $g$, it won't see which one has lower values in the other entries.

## 3.3 Greedy Algorithm

It is a simple algorithm: add the vertices which minimize the Greedy Criteria $g$ and does not violate the Precedence-constraint (Equation (3)) iteratively until no more vertex can be added without violating the Capacity-constraint (Equation (2)). Such algorithm is presented in Algorithm 1.

**Algorithm 1** Greedy

**Require:** $V, E, w, W$ ▷ This is wrong. Talk about this later, after explaining GRASP

$S \leftarrow \emptyset$
$X \leftarrow$ all root vertices
$Y \leftarrow 0 \in \mathbb{Z}^{n_w}$
$v \leftarrow \arg\min_{v \in X} g(v)$
**while** $Y + w_v \leqslant W$ **do**
    $S \leftarrow S \cup \{v\}$
    $Y \leftarrow Y + w_v$
    $X \leftarrow$ all the vertices not yet in $S$ that can be added to $S$ without violating the constraints
    $v \leftarrow \arg\min_{v \in X} \|w_v\|$
**return** $S$

We expect Algorithm 1 to give reasonable results very quickly, i.e. to be good for obtaining reasonable results but not near-optimal ones.

### 3.4 GRASP + Tabu Search

In order to find a time optimal algorithm to solve MUPKP, we propose a comparative analysis of an implementation of a Integer Linear Program to solve MPKP and a near optimal algorithm utilizing GRASP and Tabu Search as GRASP's local search.

**Algorithm 2** GRASP Pseudocode

**Require:** $MaxIterations, Seed$

**for** $k = 1, \ldots, MaxIterations$ **do**
    $Solution \leftarrow GreedyRandomizedConstruction(Seed)$;
    **if** Solution is not feasible **then**
        $Solution \leftarrow Repair(Solution)$;
    $Solution \leftarrow LocalSearch(Solution)$;
    $UpdateSolution(Solution, BestSolution)$;
**return** $BestSolution$

where the candidate list is build greedily utilizing the current maximal elements on our input graph(as long as they fit). There's no need to use any repair method as the solution built from the candidate list will always fit in the knapsack. That way, we do a local search and update our solution with the best one seen to far.

The trick here is to use a Tabu Search, instead of a naive local search method, to both avoid falling in local optima and utilizing the tabu list in order to avoid repeating moves and other. We use GRASP as a diversification strategy.

We now define exactly how this search is proposed.

Let $n \in \mathbb{N}$ be the number of binary variables of an input instance. The tabu list, let's call it $Tabu$ is a mapping $Tabu : StrN \to \mathbb{N}$, with pre-defined $Tabu.size$ capacity, where $StrN$ is an arrangement of $n$ bits let's call it $Arr$, where $n$ is the number of binary variables of our instance, and the $i$-th bit represents the state of $x_i$, where $Arr[i] = x_i$.

We also utilize a heap structure to get the earliest added tabu in $O(1)$ time.

During the local search, at each iteration we add the current state of our binary variables an element of $Tabu$. If the total number of tabus is equal to $Tabu.size$ we pop the earliest added tabu from our heap and remove that tabu from $Tabu$ while adding the new move to $Tabu$.

Each time that the Tabu Search tries to do a tabu move the integer in $Tabu$ associated with this move is looked up by the search, if the tabu exists, and if it equals 1, that tabu is removed from the mapping.

Each move in the Tabu Search is a bit flip operation of a variable in a randomly selected, at every search iteration, subset $BinVars$ of the set of all binary variables of the input instance, such that $|BinVars| = k <= n$, where $k \in \mathbb{N}$ is a pre-defined parameter, as a way to limit the search space of the Tabu Search.

The best values for $k$ and $Tabu.size$ will be determined by tests.

# 4 Instance Generation

The instances are generated randomly [9], [1], [3]. For that, first the graph is generated, and then the weight of each vertex is chosen. The knapsack capacity is selected so that, on average, X percent of the vertices fit in it. The following subsections analyze each of those aspects.

Consider the parameters:

1. $n$: number of vertices;
2. $K$: average number of branches;
3. $L$: maximum number of leaf vertices;
4. $H$: the maximum value of an entry of the weight of each vertex;
5. $m$: fraction of the average number of elements that fit in the knapsack;

## 4.1 How to Generate the Precedences

The process of generating the precedences is specified in Algorithm 3, which uses Algorithm 4. The Figure 2 has an example of such procedure. The following parameters are used to control the generation:

**Algorithm 3** Find-Trees

**Require:** $V$: vertices in the 2D plane, $K$: average number of branches, $L$: maximum number of leaf vertices

$k \leftarrow$ random number from 1 to $K$
$\langle R, \mathcal{V} \rangle \leftarrow$ find $k$ clusters in $V$ ▷ $R$: a set of centers
▷ $\mathcal{V}$: a set with each element being the set vertices of each cluster
$\mathcal{T} \leftarrow \emptyset$
**for** each pair $r \in R$ and $V' \in \mathcal{V}$ **do**
  **if** $|V'| \leqslant L$ **then**
    $T \leftarrow$ tree with $r$ as the root node and $V'$ as the leaves
  **else**
    $T \leftarrow$ tree with $r$ as the root node of the subtree Find-Trees$(V', K, L)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
**return** $\mathcal{T}$

**Algorithm 4** Generate-Precedences

**Require:** $n$: number of vertices, $K$: average number of branches, $L$: maximum number of leaf vertices

1: $V \leftarrow$ generate $n$ points in the 2D plane randomly
2: $\mathcal{T} \leftarrow$ Find-Trees$(V, K, L)$
3: **return** $\mathcal{T}$

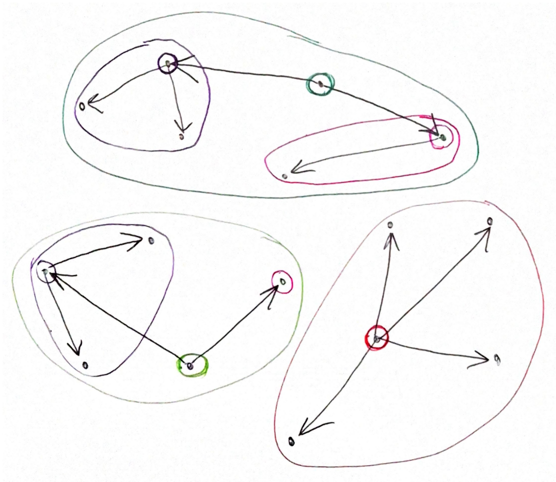

Figure 2: Precedence generation. The root nodes are the green, red and lemon. Red has four leaf vertices. Green has two branches, the pink with one leaf and the purple with two leaves. Lemon has one leaf and one branch with two leaves.

### 4.2 How to Generate the Weights

Generate the weights randomly in the interval $[0, H]$.

### 4.3 How to Generate the Knapsack Capacity

Generate each entry of the knapsack capacity $W$ randomly in the interval $[0, m \cdot n \cdot H]$.

## 5 How to evaluate the Results

We will generate a table with the result of the experiments in the format below. Graphics are going to be created on demand as we analyze the results. Such results will provide all the information required to see how each method behaves, how different instances impact on each method, how big is the instance they can handle.

| Instance | ILP | | greedy | | metaheuristic | |
|---|---|---|---|---|---|---|
| | no. items | time | no. items | time | no. items | time |
| X | 10 | 100 | 8 | 14 | 9 | 36 |

Table 1: Results of the methods of solution. The time is given in seconds.

## 6 Results

| name | problem_info | | | ilp | | greedy | | grasp | | tabu | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | capacity | edges | nodes | cost | time[s] | cost | time[s] | cost | time[s] | cost | time[s] |
| problem | | | | | | | | | | | |
| N100_E5_W52908 | 52908 | 5 | 100 | 71 | 0.001730 | 68 | 0.004 | 69 | 0.021 | 69 | 0.022 |
| N100_E5_W37781 | 37781 | 5 | 100 | 52 | 0.001786 | 49 | 0.003 | 50 | 0.016 | 50 | 0.019 |
| N100_E5_W22866 | 22866 | 5 | 100 | 32 | 0.001528 | 30 | 0.002 | 30 | 0.010 | 30 | 0.011 |
| N100_E6_W37053 | 37053 | 6 | 100 | 54 | 0.001201 | 50 | 0.004 | 50 | 0.017 | 50 | 0.016 |
| N100_E7_W52552 | 52552 | 7 | 100 | 73 | 0.005665 | 69 | 0.004 | 71 | 0.018 | 70 | 0.021 |
| N100_E7_W22507 | 22507 | 7 | 100 | 32 | 0.001855 | 29 | 0.003 | 29 | 0.013 | 29 | 0.012 |
| N100_E10_W37507 | 37507 | 10 | 100 | 53 | 0.002411 | 48 | 0.045 | 49 | 0.103 | 50 | 0.105 |
| N100_E10_W52564 | 52564 | 10 | 100 | 72 | 0.002096 | 69 | 0.019 | 70 | 0.038 | 69 | 0.043 |
| N100_E13_W22388 | 22388 | 13 | 100 | 33 | 0.001524 | 29 | 0.007 | 31 | 0.020 | 30 | 0.024 |
| N100_E17_W37381 | 37381 | 17 | 100 | 53 | 0.001444 | 50 | 0.009 | 50 | 0.033 | 50 | 0.033 |
| N100_E17_W52142 | 52142 | 17 | 100 | 71 | 0.015285 | 69 | 0.008 | 70 | 0.032 | 69 | 0.035 |
| N100_E19_W22217 | 22217 | 19 | 100 | 32 | 0.001974 | 29 | 0.003 | 29 | 0.014 | 29 | 0.027 |
| N100_E24_W52076 | 52076 | 24 | 100 | 72 | 0.001722 | 69 | 0.008 | 70 | 0.025 | 70 | 0.030 |
| N100_E25_W37882 | 37882 | 25 | 100 | 52 | 0.001780 | 49 | 0.005 | 50 | 0.020 | 50 | 0.027 |
| N100_E25_W53106 | 53106 | 25 | 100 | 73 | 0.001405 | 69 | 0.007 | 71 | 0.031 | 71 | 0.027 |
| N100_E29_W22601 | 22601 | 29 | 100 | 32 | 0.001769 | 30 | 0.003 | 30 | 0.013 | 30 | 0.014 |
| N100_E33_W37858 | 37858 | 33 | 100 | 52 | 0.003097 | 49 | 0.004 | 50 | 0.016 | 50 | 0.023 |
| N100_E34_W22399 | 22399 | 34 | 100 | 34 | 0.002070 | 30 | 0.003 | 32 | 0.013 | 32 | 0.015 |
| N100_E40_W22210 | 22210 | 40 | 100 | 32 | 0.002668 | 29 | 0.002 | 30 | 0.011 | 30 | 0.014 |
| N100_E43_W37647 | 37647 | 43 | 100 | 52 | 0.002118 | 48 | 0.004 | 49 | 0.014 | 49 | 0.024 |
| N100_E45_W22215 | 22215 | 45 | 100 | 32 | 0.001847 | 28 | 0.002 | 31 | 0.010 | 31 | 0.015 |
| N100_E46_W52707 | 52707 | 46 | 100 | 72 | 0.002069 | 69 | 0.004 | 70 | 0.021 | 70 | 0.029 |
| N100_E47_W22524 | 22524 | 47 | 100 | 33 | 0.001535 | 30 | 0.003 | 31 | 0.011 | 31 | 0.012 |
| N100_E49_W53079 | 53079 | 49 | 100 | 73 | 0.001685 | 70 | 0.007 | 72 | 0.028 | 71 | 0.025 |
| N100_E50_W52513 | 52513 | 50 | 100 | 71 | 0.002020 | 70 | 0.007 | 70 | 0.021 | 70 | 0.026 |
| N100_E51_W38089 | 38089 | 51 | 100 | 52 | 0.002069 | 48 | 0.003 | 50 | 0.015 | 50 | 0.020 |
| N100_E57_W37379 | 37379 | 57 | 100 | 53 | 0.001759 | 51 | 0.005 | 51 | 0.016 | 51 | 0.022 |

Table 2: Cost and running time of all metaheuristics for problem instances with 100 nodes.

| name | problem_info | | | ilp | | greedy | | grasp | | tabu | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | capacity | edges | nodes | cost | time[s] | cost | time[s] | cost | time[s] | cost | time[s] |
| problem | | | | | | | | | | | |
| N300_E35_W66932 | 66932 | 35 | 300 | 102 | 0.003012 | 89 | 0.012 | 92 | 0.137 | 91 | 0.115 |
| N300_E37_W111737 | 111737 | 37 | 300 | 157 | 0.004851 | 148 | 0.018 | 149 | 0.299 | 149 | 0.201 |
| N300_E41_W158233 | 158233 | 41 | 300 | 217 | 0.003906 | 211 | 0.027 | 211 | 0.299 | 210 | 0.255 |
| N300_E42_W112553 | 112553 | 42 | 300 | 162 | 0.003392 | 152 | 0.016 | 152 | 0.255 | 152 | 0.163 |
| N300_E43_W157321 | 157321 | 43 | 300 | 221 | 0.003424 | 209 | 0.019 | 212 | 0.224 | 211 | 0.200 |
| N300_E44_W113112 | 113112 | 44 | 300 | 159 | 0.004555 | 149 | 0.016 | 152 | 0.183 | 151 | 0.176 |
| N300_E46_W67993 | 67993 | 46 | 300 | 98 | 0.003879 | 89 | 0.010 | 91 | 0.110 | 91 | 0.112 |
| N300_E48_W67303 | 67303 | 48 | 300 | 97 | 0.005244 | 87 | 0.010 | 90 | 0.115 | 90 | 0.155 |
| N300_E63_W156665 | 156665 | 63 | 300 | 216 | 0.004996 | 210 | 0.022 | 211 | 0.255 | 210 | 0.240 |
| N300_E209_W158073 | 158073 | 209 | 300 | 218 | 0.004060 | 213 | 0.052 | 216 | 0.463 | 215 | 0.366 |
| N300_E218_W67693 | 67693 | 218 | 300 | 96 | 0.006152 | 91 | 0.017 | 92 | 0.138 | 92 | 0.170 |
| N300_E221_W156819 | 156819 | 221 | 300 | 215 | 0.011873 | 209 | 0.032 | 211 | 0.329 | 210 | 0.367 |
| N300_E227_W111874 | 111874 | 227 | 300 | 156 | 0.009753 | 149 | 0.019 | 151 | 0.250 | 152 | 0.258 |
| N300_E231_W158231 | 158231 | 231 | 300 | 216 | 0.005106 | 212 | 0.028 | 212 | 0.463 | 212 | 0.298 |
| N300_E232_W67504 | 67504 | 232 | 300 | 95 | 0.007596 | 90 | 0.012 | 91 | 0.143 | 91 | 0.128 |
| N300_E233_W67995 | 67995 | 233 | 300 | 99 | 0.007226 | 92 | 0.013 | 94 | 0.184 | 94 | 0.182 |
| N300_E237_W113226 | 113226 | 237 | 300 | 158 | 0.006041 | 152 | 0.034 | 153 | 0.376 | 152 | 0.244 |
| N300_E253_W112148 | 112148 | 253 | 300 | 160 | 0.007474 | 153 | 0.028 | 155 | 0.453 | 154 | 0.222 |
| N300_E393_W112377 | 112377 | 393 | 300 | 156 | 0.009241 | 152 | 0.025 | 153 | 0.307 | 153 | 0.289 |
| N300_E426_W67713 | 67713 | 426 | 300 | 94 | 0.010938 | 90 | 0.007 | 90 | 0.101 | 91 | 0.142 |
| N300_E439_W155941 | 155941 | 439 | 300 | 215 | 0.010439 | 212 | 0.033 | 213 | 0.265 | 212 | 0.284 |
| N300_E439_W114108 | 114108 | 439 | 300 | 156 | 0.008639 | 153 | 0.030 | 153 | 0.304 | 153 | 0.256 |
| N300_E443_W67549 | 67549 | 443 | 300 | 96 | 0.011604 | 92 | 0.025 | 93 | 0.141 | 93 | 0.164 |
| N300_E445_W157262 | 157262 | 445 | 300 | 214 | 0.012034 | 209 | 0.021 | 211 | 0.182 | 211 | 0.290 |
| N300_E447_W157644 | 157644 | 447 | 300 | 217 | 0.006738 | 214 | 0.034 | 215 | 0.291 | 214 | 0.286 |
| N300_E451_W112128 | 112128 | 451 | 300 | 155 | 0.069517 | 148 | 0.016 | 150 | 0.146 | 152 | 0.237 |
| N300_E466_W67544 | 67544 | 466 | 300 | 97 | 0.008810 | 93 | 0.017 | 94 | 0.142 | 94 | 0.165 |

Table 3: Cost and running time of all metaheuristics for problem instances with 300 nodes.

| name | problem_info | | | ilp | | greedy | | grasp | | tabu | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | capacity | edges | nodes | cost | time[s] | cost | time[s] | cost | time[s] | cost | time[s] |
| problem | | | | | | | | | | | |
| N500_E107_W187626 | 187626 | 107 | 500 | 265 | 0.007058 | 253 | 0.070 | 253 | 0.946 | 251 | 0.656 |
| N500_E118_W187152 | 187152 | 118 | 500 | 263 | 0.010825 | 251 | 0.059 | 252 | 0.914 | 252 | 0.678 |
| N500_E120_W187056 | 187056 | 120 | 500 | 271 | 0.006221 | 255 | 0.067 | 255 | 0.906 | 255 | 0.590 |
| N500_E121_W262283 | 262283 | 121 | 500 | 363 | 0.007442 | 352 | 0.065 | 354 | 1.194 | 352 | 0.792 |
| N500_E122_W261374 | 261374 | 122 | 500 | 360 | 0.009279 | 350 | 0.097 | 352 | 1.231 | 351 | 0.873 |
| N500_E123_W112510 | 112510 | 123 | 500 | 162 | 0.009842 | 149 | 0.024 | 151 | 0.564 | 152 | 0.475 |
| N500_E127_W113110 | 113110 | 127 | 500 | 168 | 0.008348 | 152 | 0.026 | 157 | 0.505 | 156 | 0.380 |
| N500_E130_W264094 | 264094 | 130 | 500 | 367 | 0.005548 | 355 | 0.061 | 357 | 1.338 | 357 | 0.748 |
| N500_E133_W113189 | 113189 | 133 | 500 | 164 | 0.007989 | 151 | 0.033 | 152 | 0.439 | 152 | 0.412 |
| N500_E594_W112478 | 112478 | 594 | 500 | 161 | 0.014067 | 151 | 0.024 | 155 | 0.706 | 155 | 0.577 |
| N500_E594_W112090 | 112090 | 594 | 500 | 158 | 1.634213 | 151 | 0.026 | 153 | 0.552 | 153 | 0.566 |
| N500_E614_W258557 | 258557 | 614 | 500 | 360 | 0.011334 | 354 | 0.106 | 355 | 1.555 | 354 | 1.046 |
| N500_E621_W261167 | 261167 | 621 | 500 | 360 | 0.010790 | 353 | 0.103 | 355 | 1.338 | 353 | 1.190 |
| N500_E621_W186642 | 186642 | 621 | 500 | 260 | 0.014489 | 251 | 0.052 | 253 | 1.079 | 254 | 1.033 |
| N500_E623_W111165 | 111165 | 623 | 500 | 164 | 0.011128 | 154 | 0.056 | 157 | 0.719 | 157 | 0.609 |
| N500_E626_W261199 | 261199 | 626 | 500 | 357 | 0.056487 | 350 | 0.065 | 351 | 1.225 | 353 | 0.969 |
| N500_E629_W189219 | 189219 | 629 | 500 | 266 | 0.010908 | 256 | 0.138 | 259 | 1.735 | 259 | 1.322 |
| N500_E650_W187605 | 187605 | 650 | 500 | 261 | 0.013607 | 255 | 0.104 | 255 | 1.311 | 254 | 1.048 |
| N500_E1154_W188267 | 188267 | 1154 | 500 | 257 | 0.068704 | 253 | 0.031 | 253 | 0.396 | 252 | 0.902 |
| N500_E1155_W112554 | 112554 | 1155 | 500 | 158 | 0.035928 | 152 | 0.027 | 154 | 0.512 | 153 | 0.607 |
| N500_E1160_W262039 | 262039 | 1160 | 500 | 359 | 0.024286 | 355 | 0.051 | 357 | 0.886 | 353 | 1.019 |
| N500_E1169_W263485 | 263485 | 1169 | 500 | 357 | 0.028345 | 353 | 0.044 | 353 | 0.658 | 352 | 1.089 |
| N500_E1172_W112045 | 112045 | 1172 | 500 | 156 | 0.039502 | 151 | 0.021 | 151 | 0.278 | 152 | 0.565 |
| N500_E1175_W111948 | 111948 | 1175 | 500 | 159 | 0.024668 | 155 | 0.036 | 156 | 0.408 | 156 | 0.547 |
| N500_E1176_W187733 | 187733 | 1176 | 500 | 257 | 0.028575 | 251 | 0.025 | 254 | 0.675 | 252 | 0.923 |
| N500_E1192_W187394 | 187394 | 1192 | 500 | 259 | 0.074943 | 253 | 0.040 | 256 | 0.608 | 255 | 0.917 |
| N500_E1203_W263300 | 263300 | 1203 | 500 | 356 | 0.074708 | 350 | 0.023 | 351 | 0.393 | 350 | 1.031 |

Table 4: Cost and running time of all metaheuristics for problem instances with 500 nodes.

## References


[1] Byungjun You and Takeo Yamada. A pegging approach to the precedence-constrained knapsack problem. *European journal of operational research*, 183(2):618–632, 2007.

[2] Cezar A Mortari. *Introdução à lógica*. Unesp, 2001.

[3] Zhen Yang, Guoqing Wang, and Feng Chu. An effective grasp and tabu search for the 0–1 quadratic knapsack problem. *Computers & Operations Research*, 40(5):1176–1185, 2013.

[4] Said Hanafi and Arnaud Freville. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2-3):659–675, 1998.

[5] A Freville and G Plateau. *Méthodes heuristiques performantes pour les problèmes en variables 0-1 à plusieurs contraintes en inégalité*. Université de Lille I, UER d'IEEA Informatique, 1982.

[6] Arnaud Freville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete applied mathematics*, 49(1-3):189–212, 1994.

[7] Fred Glover and Gary A Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Meta-heuristics*, pages 407–427. Springer, 1996.

[8] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[9] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.