Project - Multidimensional Unitary-profit Precedence-constrained Knapsack Problem

Luiz Fernando Bueno Rosa - RA: 221197
Lucas Guesser Targino da Silva - RA: 203534

July 15, 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. For cases in which the exact solution is not available, a Duality Gap is used to compute the optimality gap.

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.

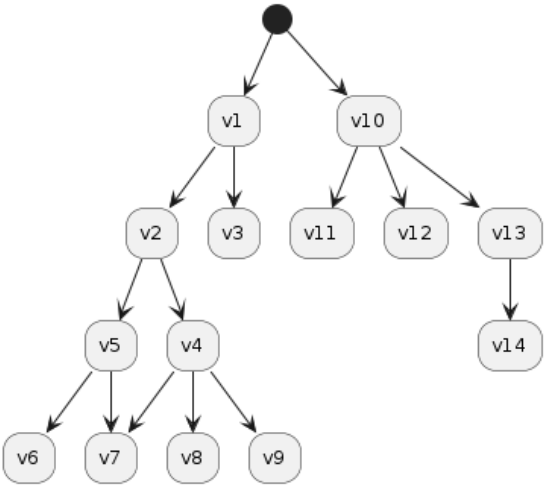

Figure 1: Example of a directed acyclic graph. The black dot indicates the root vertices. For this case, the induced partial order satisfy: $v5 \prec v2$, $v7 \prec v1$, $v14 \prec v10$.

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) is the maximum weight constraint, 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

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

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.

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 subgradient method to solve the problem faster then use a "pegging" test to guarantee optimality.

2.1 0-1 Knapsack Problem

In [8], the authors give a definition for a 0-1 Knapsack Problem (0-1KP):

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

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

3.1 Integer Linear Programming Model

3.1.1 Decision Variables

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

3.1.2 Mathematical Model

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

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

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

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

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.

3.2 GRASP + Tabu Search

In order to find a time optimal algorithm to solve the multidimensional precedence-constrained knapsack problem (MPKP), 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 1 GRASP Pseudocode

Require: $MaxIterations, Seed$

```
for k = 1, ..., MaxIterations do
    Solution ← GreedyRandomizedConstruction(Seed);
    if Solution is not feasible then
        Solution ← Repair(Solution);
    Solution ← 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.

3.3 Greedy Algorithm

It is a simple algorithm: add the vertices which minimize the added weight and does not violate the precedence-constraint iteratively until no more vertex can be added without exceeding the knapsack capacity. Such algorithm is presented below:

Algorithm 2 Greedy

Require: V, E, w, W

1: $S \leftarrow \emptyset$
2: $X \leftarrow$ all leaf vertices
3: $Y \leftarrow 0$
4: $v \leftarrow \arg\min_{v \in X} w_v$
5: **while** $Y + w_v \leqslant W$ **do**
6: $\quad S \leftarrow S \cup \{v\}$
7: $\quad Y \leftarrow Y + w_v$
8: $\quad X \leftarrow$ all the vertices not yet in S that can be added to S without violating the constraints
9: $\quad v \leftarrow \arg\min_{v \in X} \|w_v\|$
10: **return** S

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:

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 $[0, m \cdot n \cdot H]$.

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

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

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

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.

	problems	cost	running time [s]
0	N100_E10_W37507	53	0.002411
1	N100_E10_W52564	72	0.002096
2	N100_E13_W22388	33	0.001524
3	N100_E17_W37381	53	0.001444
4	N100_E17_W52142	71	0.015285
5	N100_E19_W22217	32	0.001974
6	N100_E24_W52076	72	0.001722
7	N100_E25_W37882	52	0.001780
8	N100_E25_W53106	73	0.001405
9	N100_E29_W22601	32	0.001769
10	N100_E33_W37858	52	0.003097
11	N100_E34_W22399	34	0.002070
12	N100_E40_W22210	32	0.002668
13	N100_E43_W37647	52	0.002118
14	N100_E45_W22215	32	0.001847
15	N100_E46_W52707	72	0.002069
16	N100_E47_W22524	33	0.001535
17	N100_E49_W53079	73	0.001685
18	N100_E50_W52513	71	0.002020
19	N100_E51_W38089	52	0.002069
20	N100_E57_W37379	53	0.001759
21	N100_E5_W22866	32	0.001528
22	N100_E5_W37781	52	0.001786
23	N100_E5_W52908	71	0.001730
24	N100_E6_W37053	54	0.001201
25	N100_E7_W22507	32	0.001855
26	N100_E7_W52552	73	0.005665

Table 2: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
27	N300_E209_W158073	218	0.004060
28	N300_E218_W67693	96	0.006152
29	N300_E221_W156819	215	0.011873
30	N300_E227_W111874	156	0.009753
31	N300_E231_W158231	216	0.005106
32	N300_E232_W67504	95	0.007596
33	N300_E233_W67995	99	0.007226
34	N300_E237_W113226	158	0.006041
35	N300_E253_W112148	160	0.007474
36	N300_E35_W66932	102	0.003012
37	N300_E37_W111737	157	0.004851
38	N300_E393_W112377	156	0.009241
39	N300_E41_W158233	217	0.003906
40	N300_E426_W67713	94	0.010938
41	N300_E42_W112553	162	0.003392
42	N300_E439_W114108	156	0.008639
43	N300_E439_W155941	215	0.010439
44	N300_E43_W157321	221	0.003424
45	N300_E443_W67549	96	0.011604
46	N300_E445_W157262	214	0.012034
47	N300_E447_W157644	217	0.006738
48	N300_E44_W113112	159	0.004555
49	N300_E451_W112128	155	0.069517
50	N300_E466_W67544	97	0.008810
51	N300_E46_W67993	98	0.003879
52	N300_E48_W67303	97	0.005244
53	N300_E63_W156665	216	0.004996

Table 3: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
54	N500_E107_W187626	265	0.007058
55	N500_E1154_W188267	257	0.068704
56	N500_E1155_W112554	158	0.035928
57	N500_E1160_W262039	359	0.024286
58	N500_E1169_W263485	357	0.028345
59	N500_E1172_W112045	156	0.039502
60	N500_E1175_W111948	159	0.024668
61	N500_E1176_W187733	257	0.028575
62	N500_E118_W187152	263	0.010825
63	N500_E1192_W187394	259	0.074943
64	N500_E1203_W263300	356	0.074708
65	N500_E120_W187056	271	0.006221
66	N500_E121_W262283	363	0.007442
67	N500_E122_W261374	360	0.009279
68	N500_E123_W112510	162	0.009842
69	N500_E127_W113110	168	0.008348
70	N500_E130_W264094	367	0.005548
71	N500_E133_W113189	164	0.007989
72	N500_E594_W112090	158	1.634213
73	N500_E594_W112478	161	0.014067
74	N500_E614_W258557	360	0.011334
75	N500_E621_W186642	260	0.014489
76	N500_E621_W261167	360	0.010790
77	N500_E623_W111165	164	0.011128
78	N500_E626_W261199	357	0.056487
79	N500_E629_W189219	266	0.010908
80	N500_E650_W187605	261	0.013607

Table 4: Cost and running time of the ilp metaheuristic for several problems.

	problems	cost	running time [s]
0	N100_E10_W37507	48	0.045
1	N100_E10_W52564	69	0.019
2	N100_E13_W22388	29	0.007
3	N100_E17_W37381	50	0.009
4	N100_E17_W52142	69	0.008
5	N100_E19_W22217	29	0.003
6	N100_E24_W52076	69	0.008
7	N100_E25_W37882	49	0.005
8	N100_E25_W53106	69	0.007
9	N100_E29_W22601	30	0.003
10	N100_E33_W37858	49	0.004
11	N100_E34_W22399	30	0.003
12	N100_E40_W22210	29	0.002
13	N100_E43_W37647	48	0.004
14	N100_E45_W22215	28	0.002
15	N100_E46_W52707	69	0.004
16	N100_E47_W22524	30	0.003
17	N100_E49_W53079	70	0.007
18	N100_E50_W52513	70	0.007
19	N100_E51_W38089	48	0.003
20	N100_E57_W37379	51	0.005
21	N100_E5_W22866	30	0.002
22	N100_E5_W37781	49	0.003
23	N100_E5_W52908	68	0.004
24	N100_E6_W37053	50	0.004
25	N100_E7_W22507	29	0.003
26	N100_E7_W52552	69	0.004

Table 5: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
27	N300_E209_W158073	213	0.052
28	N300_E218_W67693	91	0.017
29	N300_E221_W156819	209	0.032
30	N300_E227_W111874	149	0.019
31	N300_E231_W158231	212	0.028
32	N300_E232_W67504	90	0.012
33	N300_E233_W67995	92	0.013
34	N300_E237_W113226	152	0.034
35	N300_E253_W112148	153	0.028
36	N300_E35_W66932	89	0.012
37	N300_E37_W111737	148	0.018
38	N300_E393_W112377	152	0.025
39	N300_E41_W158233	211	0.027
40	N300_E426_W67713	90	0.007
41	N300_E42_W112553	152	0.016
42	N300_E439_W114108	153	0.030
43	N300_E439_W155941	212	0.033
44	N300_E43_W157321	209	0.019
45	N300_E443_W67549	92	0.025
46	N300_E445_W157262	209	0.021
47	N300_E447_W157644	214	0.034
48	N300_E44_W113112	149	0.016
49	N300_E451_W112128	148	0.016
50	N300_E466_W67544	93	0.017
51	N300_E46_W67993	89	0.010
52	N300_E48_W67303	87	0.010
53	N300_E63_W156665	210	0.022

Table 6: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
54	N500_E107_W187626	253	0.070
55	N500_E1154_W188267	253	0.031
56	N500_E1155_W112554	152	0.027
57	N500_E1160_W262039	355	0.051
58	N500_E1169_W263485	353	0.044
59	N500_E1172_W112045	151	0.021
60	N500_E1175_W111948	155	0.036
61	N500_E1176_W187733	251	0.025
62	N500_E118_W187152	251	0.059
63	N500_E1192_W187394	253	0.040
64	N500_E1203_W263300	350	0.023
65	N500_E120_W187056	255	0.067
66	N500_E121_W262283	352	0.065
67	N500_E122_W261374	350	0.097
68	N500_E123_W112510	149	0.024
69	N500_E127_W113110	152	0.026
70	N500_E130_W264094	355	0.061
71	N500_E133_W113189	151	0.033
72	N500_E594_W112090	151	0.026
73	N500_E594_W112478	151	0.024
74	N500_E614_W258557	354	0.106
75	N500_E621_W186642	251	0.052
76	N500_E621_W261167	353	0.103
77	N500_E623_W111165	154	0.056
78	N500_E626_W261199	350	0.065
79	N500_E629_W189219	256	0.138
80	N500_E650_W187605	255	0.104

Table 7: Cost and running time of the greedy metaheuristic for several problems.

	problems	cost	running time [s]
0	N100_E10_W37507	49	0.103
1	N100_E10_W52564	70	0.038
2	N100_E13_W22388	31	0.020
3	N100_E17_W37381	50	0.033
4	N100_E17_W52142	70	0.032
5	N100_E19_W22217	29	0.014
6	N100_E24_W52076	70	0.025
7	N100_E25_W37882	50	0.020
8	N100_E25_W53106	71	0.031
9	N100_E29_W22601	30	0.013
10	N100_E33_W37858	50	0.016
11	N100_E34_W22399	32	0.013
12	N100_E40_W22210	30	0.011
13	N100_E43_W37647	49	0.014
14	N100_E45_W22215	31	0.010
15	N100_E46_W52707	70	0.021
16	N100_E47_W22524	31	0.011
17	N100_E49_W53079	72	0.028
18	N100_E50_W52513	70	0.021
19	N100_E51_W38089	50	0.015
20	N100_E57_W37379	51	0.016
21	N100_E5_W22866	30	0.010
22	N100_E5_W37781	50	0.016
23	N100_E5_W52908	69	0.021
24	N100_E6_W37053	50	0.017
25	N100_E7_W22507	29	0.013
26	N100_E7_W52552	71	0.018

Table 8: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
27	N300_E209_W158073	216	0.463
28	N300_E218_W67693	92	0.138
29	N300_E221_W156819	211	0.329
30	N300_E227_W111874	151	0.250
31	N300_E231_W158231	212	0.463
32	N300_E232_W67504	91	0.143
33	N300_E233_W67995	94	0.184
34	N300_E237_W113226	153	0.376
35	N300_E253_W112148	155	0.453
36	N300_E35_W66932	92	0.137
37	N300_E37_W111737	149	0.299
38	N300_E393_W112377	153	0.307
39	N300_E41_W158233	211	0.299
40	N300_E426_W67713	90	0.101
41	N300_E42_W112553	152	0.255
42	N300_E439_W114108	153	0.304
43	N300_E439_W155941	213	0.265
44	N300_E43_W157321	212	0.224
45	N300_E443_W67549	93	0.141
46	N300_E445_W157262	211	0.182
47	N300_E447_W157644	215	0.291
48	N300_E44_W113112	152	0.183
49	N300_E451_W112128	150	0.146
50	N300_E466_W67544	94	0.142
51	N300_E46_W67993	91	0.110
52	N300_E48_W67303	90	0.115
53	N300_E63_W156665	211	0.255

Table 9: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
54	N500_E107_W187626	253	0.946
55	N500_E1154_W188267	253	0.396
56	N500_E1155_W112554	154	0.512
57	N500_E1160_W262039	357	0.886
58	N500_E1169_W263485	353	0.658
59	N500_E1172_W112045	151	0.278
60	N500_E1175_W111948	156	0.408
61	N500_E1176_W187733	254	0.675
62	N500_E118_W187152	252	0.914
63	N500_E1192_W187394	256	0.608
64	N500_E1203_W263300	351	0.393
65	N500_E120_W187056	255	0.906
66	N500_E121_W262283	354	1.194
67	N500_E122_W261374	352	1.231
68	N500_E123_W112510	151	0.564
69	N500_E127_W113110	157	0.505
70	N500_E130_W264094	357	1.338
71	N500_E133_W113189	152	0.439
72	N500_E594_W112090	153	0.552
73	N500_E594_W112478	155	0.706
74	N500_E614_W258557	355	1.555
75	N500_E621_W186642	253	1.079
76	N500_E621_W261167	355	1.338
77	N500_E623_W111165	157	0.719
78	N500_E626_W261199	351	1.225
79	N500_E629_W189219	259	1.735
80	N500_E650_W187605	255	1.311

Table 10: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
0	N100_E10_W37507	50	0.105
1	N100_E10_W52564	69	0.043
2	N100_E13_W22388	30	0.024
3	N100_E17_W37381	50	0.033
4	N100_E17_W52142	69	0.035
5	N100_E19_W22217	29	0.027
6	N100_E24_W52076	70	0.030
7	N100_E25_W37882	50	0.027
8	N100_E25_W53106	71	0.027
9	N100_E29_W22601	30	0.014
10	N100_E33_W37858	50	0.023
11	N100_E34_W22399	32	0.015
12	N100_E40_W22210	30	0.014
13	N100_E43_W37647	49	0.024
14	N100_E45_W22215	31	0.015
15	N100_E46_W52707	70	0.029
16	N100_E47_W22524	31	0.012
17	N100_E49_W53079	71	0.025
18	N100_E50_W52513	70	0.026
19	N100_E51_W38089	50	0.020
20	N100_E57_W37379	51	0.022
21	N100_E5_W22866	30	0.011
22	N100_E5_W37781	50	0.019
23	N100_E5_W52908	69	0.022
24	N100_E6_W37053	50	0.016
25	N100_E7_W22507	29	0.012
26	N100_E7_W52552	70	0.021

Table 11: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
27	N300_E209_W158073	215	0.366
28	N300_E218_W67693	92	0.170
29	N300_E221_W156819	210	0.367
30	N300_E227_W111874	152	0.258
31	N300_E231_W158231	212	0.298
32	N300_E232_W67504	91	0.128
33	N300_E233_W67995	94	0.182
34	N300_E237_W113226	152	0.244
35	N300_E253_W112148	154	0.222
36	N300_E35_W66932	91	0.115
37	N300_E37_W111737	149	0.201
38	N300_E393_W112377	153	0.289
39	N300_E41_W158233	210	0.255
40	N300_E426_W67713	91	0.142
41	N300_E42_W112553	152	0.163
42	N300_E439_W114108	153	0.256
43	N300_E439_W155941	212	0.284
44	N300_E43_W157321	211	0.200
45	N300_E443_W67549	93	0.164
46	N300_E445_W157262	211	0.290
47	N300_E447_W157644	214	0.286
48	N300_E44_W113112	151	0.176
49	N300_E451_W112128	152	0.237
50	N300_E466_W67544	94	0.165
51	N300_E46_W67993	91	0.112
52	N300_E48_W67303	90	0.155
53	N300_E63_W156665	210	0.240

Table 12: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
54	N500_E107_W187626	251	0.656
55	N500_E1154_W188267	252	0.902
56	N500_E1155_W112554	153	0.607
57	N500_E1160_W262039	353	1.019
58	N500_E1169_W263485	352	1.089
59	N500_E1172_W112045	152	0.565
60	N500_E1175_W111948	156	0.547
61	N500_E1176_W187733	252	0.923
62	N500_E118_W187152	252	0.678
63	N500_E1192_W187394	255	0.917
64	N500_E1203_W263300	350	1.031
65	N500_E120_W187056	255	0.590
66	N500_E121_W262283	352	0.792
67	N500_E122_W261374	351	0.873
68	N500_E123_W112510	152	0.475
69	N500_E127_W113110	156	0.380
70	N500_E130_W264094	357	0.748
71	N500_E133_W113189	152	0.412
72	N500_E594_W112090	153	0.566
73	N500_E594_W112478	155	0.577
74	N500_E614_W258557	354	1.046
75	N500_E621_W186642	254	1.033
76	N500_E621_W261167	353	1.190
77	N500_E623_W111165	157	0.609
78	N500_E626_W261199	353	0.969
79	N500_E629_W189219	259	1.322
80	N500_E650_W187605	254	1.048

Table 13: Cost and running time of the tabu metaheuristic for several problems.

	problems	cost	running time [s]
0	N1000_E2405_W373757	500	0.369
1	N1000_E2413_W224743	300	0.203
2	N1000_E2414_W527164	704	0.231
3	N1000_E2417_W376515	509	0.364
4	N1000_E2418_W524518	709	0.279
5	N1000_E2422_W375622	506	0.187
6	N1000_E2426_W523016	701	0.111
7	N1000_E2438_W225461	302	0.115
8	N1000_E2445_W223283	309	0.189
9	N1000_E3542_W525104	699	0.032
10	N1000_E3578_W224047	300	0.037
11	N1000_E3624_W525657	703	0.070
12	N1000_E3637_W225015	296	0.022
13	N1000_E3666_W226147	303	0.041
14	N1000_E3668_W376329	506	0.074
15	N1000_E3681_W527379	702	0.063
16	N1000_E3704_W375910	504	0.057
17	N1000_E3737_W375969	496	0.022
18	N1000_E477_W524540	704	0.487
19	N1000_E483_W225301	302	0.114
20	N1000_E486_W376292	507	0.365
21	N1000_E493_W525172	709	0.463
22	N1000_E494_W524333	713	0.552
23	N1000_E495_W375913	506	0.379
24	N1000_E498_W374946	511	0.351
25	N1000_E522_W225006	310	0.155
26	N1000_E526_W224769	306	0.140

Table 14: Cost and running time of the greedy metaheuristic for several problems.

	problems	cost	running time [s]
0	N1000_E2405_W373757	504	4.216
1	N1000_E2413_W224743	305	3.441
2	N1000_E2414_W527164	704	4.367
3	N1000_E2417_W376515	509	6.484
4	N1000_E2418_W524518	710	7.222
5	N1000_E2422_W375622	507	5.401
6	N1000_E2426_W523016	703	3.339
7	N1000_E2438_W225461	303	2.583
8	N1000_E2445_W223283	311	4.482
9	N1000_E3542_W525104	701	0.902
10	N1000_E3578_W224047	303	1.018
11	N1000_E3624_W525657	704	1.865
12	N1000_E3637_W225015	299	0.612
13	N1000_E3666_W226147	305	1.420
14	N1000_E3668_W376329	509	1.769
15	N1000_E3681_W527379	705	2.136
16	N1000_E3704_W375910	505	1.436
17	N1000_E3737_W375969	500	0.901
18	N1000_E477_W524540	705	14.012
19	N1000_E483_W225301	304	4.133
20	N1000_E486_W376292	511	12.103
21	N1000_E493_W525172	712	16.690
22	N1000_E494_W524333	717	16.163
23	N1000_E495_W375913	506	11.066
24	N1000_E498_W374946	516	12.712
25	N1000_E522_W225006	313	5.242
26	N1000_E526_W224769	309	5.256

Table 15: Cost and running time of the grasp metaheuristic for several problems.

	problems	cost	running time [s]
0	N1000_E2405_W373757	505	6.277
1	N1000_E2413_W224743	305	3.857
2	N1000_E2414_W527164	702	6.477
3	N1000_E2417_W376515	508	6.699
4	N1000_E2418_W524518	705	7.138
5	N1000_E2422_W375622	508	6.232
6	N1000_E2426_W523016	700	5.879
7	N1000_E2438_W225461	305	3.705
8	N1000_E2445_W223283	312	4.316
9	N1000_E3542_W525104	702	5.630
10	N1000_E3578_W224047	302	2.648
11	N1000_E3624_W525657	702	5.737
12	N1000_E3637_W225015	300	2.615
13	N1000_E3666_W226147	303	2.859
14	N1000_E3668_W376329	507	4.160
15	N1000_E3681_W527379	703	5.674
16	N1000_E3704_W375910	505	4.190
17	N1000_E3737_W375969	498	4.246
18	N1000_E477_W524540	704	5.385
19	N1000_E483_W225301	306	2.705
20	N1000_E486_W376292	509	4.675
21	N1000_E493_W525172	711	5.130
22	N1000_E494_W524333	716	4.757
23	N1000_E495_W375913	506	4.496
24	N1000_E498_W374946	516	4.666
25	N1000_E522_W225006	311	2.903
26	N1000_E526_W224769	308	2.975

Table 16: Cost and running time of the tabu metaheuristic for several problems.

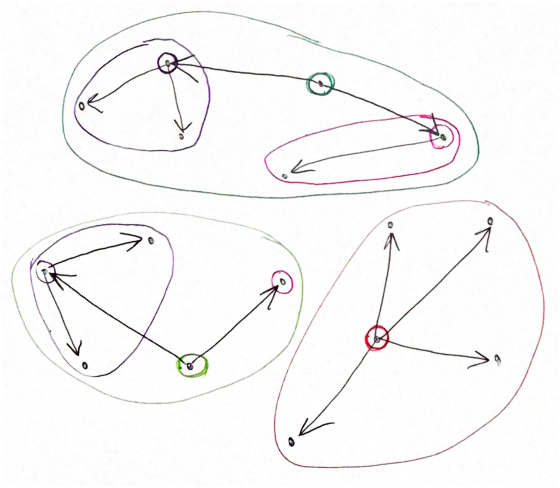

Figure 2: Precedence generation. The root nodes are the green, red and lemon. Red has four leaf vertices. Green has two branches, the pink with one leaf and the purple with two leaves. Lemon has one leaf and one branch with two leaves.

[4] 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.