Project - Multidimensional Equal-valued Precedence-constrained Knapsack Problem

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

June 3, 2022

1 checklist what to do

- Título
  - denominação do problema
  - metodologia de solução
- Resumo
  - objetivos
  - informações sobre o problema
  - metodologia de solução
  - como será feita a avaliação dos resultados
- Introdução
  - descrição formal do problema (formulação matemática)
  - revisão bibliográfica do problema (e/ou problemas relacionados)
  - metodologias previamente utilizadas
- Metodologia
  - justificativa
  - descrição das técnicas de otimização
  - descrever as técnicas de otimização contextualizando-as ao problema de otimização combinatória proposto
- Avaliação dos Resultados
  - proposta de experimentos
  - descrição das instâncias
  - como será feita a avaliação dos resultados
- Referências Bibliográficas

Abstract

We present a generalization of the classic 0-1 Knapsack Problem. First, the weight of the items are multidimensional vectors and so is the knapsack capacity. Second, the profit of all items is equal to one. Third, the items are required to be added in a certain order. That problem is reffered as *Multidimensional Equal-valued Precedence-constrained Knapsack Problem (MEPKP)*. Three solution approaches are proposed: 1 an Integer Linear Programming for a exact solution; 2 a greedy algorithm for a fast solution; 3 a Greedy Randomized Adaptive Search Procedures (GRASP) and Tabu Search (TS) for a near optimal solution; The three approaches are 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.

2 Problem Statement

2.1 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 $n_w \in \natural$;
   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;

2.1.1 Partial Order

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

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

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

2.2 Output

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

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

$$\forall v \, (v \in S \to \forall v' \, (v' \prec v \to v' \in S)) \tag{3}$$

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' lower than it (in the sense of the partial order $\prec$) must also be included. It is called Precedence-Constraint.

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

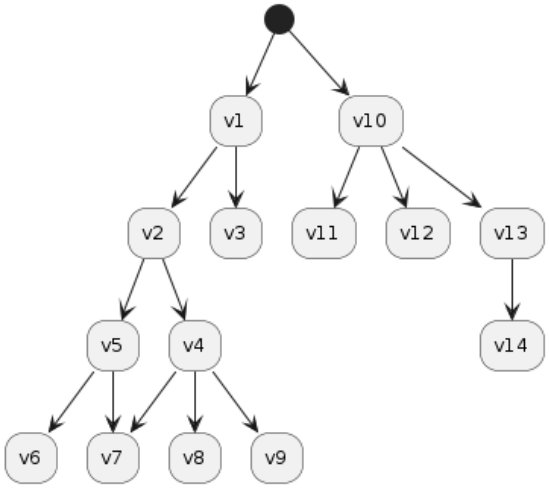

Figure 1: Example of a directed acyclic graph. The black dot indicates the root vertices. For this case, the induced partial order satisfy: $v5 \prec v2$, $v7 \prec v1$, $v14 \prec v10$.

2.3 Objective

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

3 Integer Linear Programming Model

3.1 Decision Variables

$$x_v = \begin{cases} 1 & , v \in S \\ 0 & , v \notin S \end{cases} \tag{4}$$

3.2 Mathematical Model

$$\max_{S \subseteq V} \sum_{v \in S} x_v \tag{5}$$

$$s.t. \sum_{v \in S} x_v w_v \leqslant W \tag{6}$$

$$x_v \leqslant x_{v'} \quad \forall v' \prec v \tag{7}$$

$$x \in \{0,1\}^{n_w} \tag{8}$$

Equation (5) is the objective function: maximize the number of vertices in the solution. Equation (6) is the Capacity-Constraint of Equation (2). Equation (7) is the Precedence-Constraint of Equation (3): if a vertex v is in the solution, then all vertices v' for which there is a path from v must also be in the solution.

4 Instance Generation

The instances are generated randomly [3], [1], [4]. 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 1, which uses Algorithm 2. The Figure 2 has an example of such procedure. The following parameters are used to control the generation:

Algorithm 1 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 2 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 $\mathcal{T}$

4.2 How to Generate the Weights

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

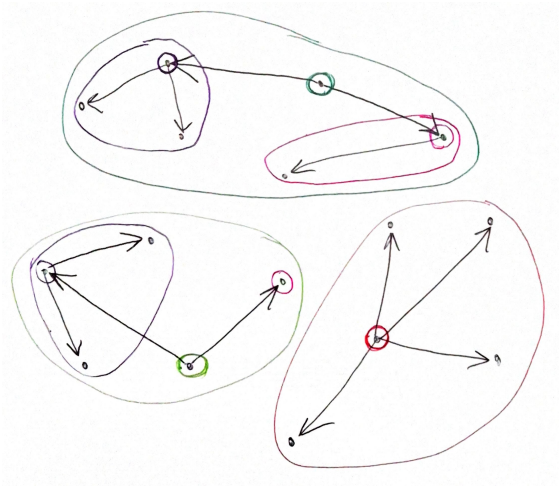

Figure 2: Precedence generation. The root nodes are the green, red and lemon. Red has four leaf vertices. Green has two branches, the pink with one leaf and the purple with two leaves. Lemon has one leaf and one branch with two leaves.

4.3 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.

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] 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.

[4] 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.