UNIVERSIDADE FEDERAL DO PARANÁ

GUSTAVO VALENTE NUNES

MODELAGEM DE UM PROBLEMA DE ENVIO UTILIZANDO PROGRAMAÇÃO LINEAR

CURITIBA PR

2022

GUSTAVO VALENTE NUNES

MODELAGEM DE UM PROBLEMA DE ENVIO UTILIZANDO PROGRAMAÇÃO LINEAR

Trabalho apresentado como requisito parcial à conclusão da disciplina de Otimização no Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

CURITIBA PR

2022

SUMÁRIO

1 INTRODUÇÃO

O problema sugerido pelo orientador consiste em modelar e implementar uma solução para o problema de envio ordenado. Existe uma certa quantidade de produtos disponíveis e esses produtos precisam ser enviados em uma quantidade finita de viagens. O objetivo do trabalho, é encontrar uma solução que minimize a quantidade de viagens necessárias para realizar o transporte de todas as mercadorias. Os itens que serão transportados podem ser separados em pedaços menores de forma que seja possível separar os pedaços em caminhões diferentes.

1.1 LIMITAÇÕES E DETALHES

Antes de começar a análise e modelagem do problema, vou definir alguns detalhes, limitações e problemas encontrados.

- O custo mínimo é calculado sobre a quantidade mínima de viagens necessárias, de forma que eu consiga fazer o transporte de todos os produtos.
- Cada caminhão possui um limite de peso que ele consegue carregar. E esse peso precisa ser respeitado pelos itens que estarão dentro desse caminhão.
- Os caminhões podem carregar diferentes itens, contanto que o peso seja respeitado.
- Na modelagem do problema, os pares ordenados não foram considerados. Isso porque não foi encontrada uma forma de se utilizar os pares ordenados utilizando da programação linear.
- Se necessário, um item pode ser separado em diferentes caminhões.

1.2 VARIÁVEIS DO PROBLEMA

Nesta seção é apresentado as variáveis e restrições que foram utilizadas na modelagem do problema.

Variáveis Gerais:

- n = Quantidade de itens que serão transportados.
- C = Capacidade total possível para cada caminhão.

Váriaveis referente ao problema:

- w_i = representa o peso do item i.
- y_j = representa o caminhão j.
- $x_{i,j}$ = representa o item i no caminhão j.

Essas são todas as variáveis necessárias para realizar a modelagem do problema. Nos próximos capítulos serão abordados a resposta ótima para o problema, a modelagem/restrições, como foi implementado e um exemplo prático.

1.3 ESCOLHA DA FERRAMENTA DE MODELAGEM

O problema de envio de cargas será resolvido utilizando as técnicas de programação linear. Nesta área, uma das principais preocupações é encontrar uma solução ótima para o problema. Neste problema, a solução ótima é aquela que respeita as restrições e o custo total mínimo. A técnica de programação linear que será utilizada, é o simplex. Iremos utilizar a linguagem de programação python junto com uma ferramenta chamada ortools do Google e como essa ferramenta nos disponibiliza vários resolvedores possíveis, o escolhido é o "GLOP".

1.3.1 Exemplo de como utilizar a ferramenta

Para facilitar a leitura dos problemas posteriores, nessa seção será apresentado um passo a passo de um problema básico, desde sua modelagem, até solução. Vamos supor o seguinte problema de programação linear:

$$max : 3x + 4y$$

$$st : x + 2y \leq 14$$

$$st : 3x - y \geq 0$$

$$st : x - y \leq 2$$

Tanto a função objetiva 3x + 4y, quanto suas restrições são representados por expressões lineares, criando assim um problema de programação linear. As restrições do problema, criam a seguinte região.

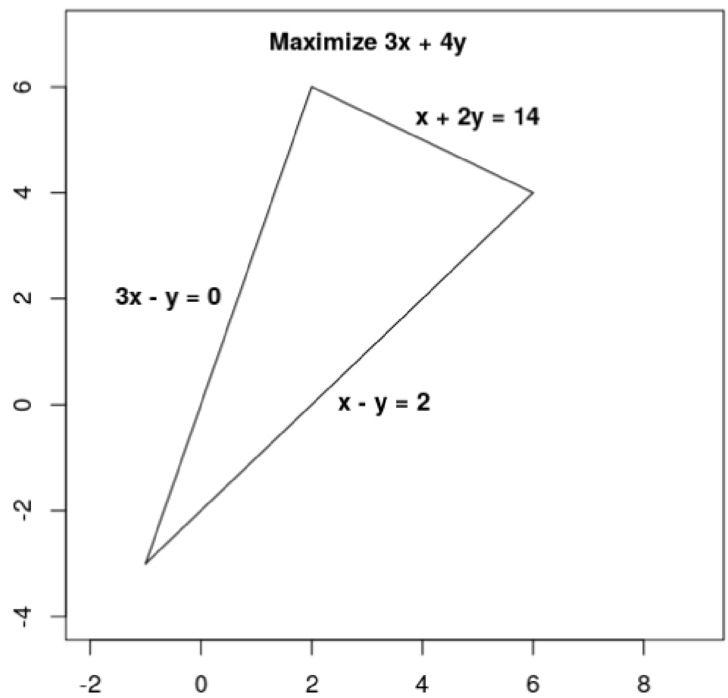

Imagem 1: Possíveis resultados.

Para começarmos a resolução do problema de exemplo, precisaremos fazer o import da biblioteca e selecionar o resolver que queremos.

```python
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver('GLOP')
```

Imagem 1: Import e Resolvedor

Como já explicado acima, o "GLOP"é um resolvedor de problemas de programação linear que utiliza o algoritmo do simplex. Agora que já temos o resolvedor pronto, precisamos criar as variáveis e as restrições. Para criar as variáveis, precisamos fazer o seguinte:

```python
x = solver.NumVar(0, solver.infinity(), 'x')
y = solver.NumVar(0, solver.infinity(), 'y')
```

Imagem 2: Variáveis do problema.

Com as váriaveis criadas, precisamos criar as restrições necessáris.

```python
solver.Add(x + 2 * y <= 14.0)
solver.Add(3 * x - y >= 0.0)
solver.Add(x - y <= 2.0)
```

Imagem 3: Restrições do problema.

Agora que temos tanto as variáveis do problema e as restrições, preciso criar minha função objetiva e chamar o resolvedor.

```python
solver.Maximize(3 * x + 4 * y)
status = solver.Solve()
```

Imagem 3: Restrições do problema.

Agora com tudo pronto, teremos a seguinte resposta.

```
Solution:
Objective value = 33.99999999999999
x = 5.999999999999998
y = 3.9999999999999996
```

Imagem 4: Restrições do problema.

Esse é um exemplo de como funciona a ferramenta or-tools, dessa mesma forma será feito a modelagem do problema de envio de cargas.

2 MODELAGEM E RESTRIÇÕES

Neste capítulo será realizado a modelagem do problema utilizando as variáveis já descritas nos capítulos anteriores. De forma a facilitar o entendimento, será apresentado a modelagem inteira de uma vez e em seguida será discutido passo a passo de como ela funciona.

Inequações	Restrições	Limite
Inequação 1	$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} = 1$	$1 \leq i, j \leq n$
Inequação 2	$\sum_{i=1}^{n} \sum_{j=1}^{n} w_i * x_{i,j} \leq C * y_i$	$1 \leq i, j \leq n$
Inequação 3	$x_{i,j} \in \{0.0, 1.0\}$	$1 \leq i, j \leq n$
Inequação 4	$y_j \in \{0.0, 1.0\}$	$1 \leq j \leq n$

INEQUAÇÃO 1: A inequação 1 nos diz a quantidade do item i que está contida dentro do caminhão j, essa soma não pode ultrapassar o 1, que indica que o item já foi completamente transportado. Essa restrição é importante, porque confirma quando o item inteiro já foi transportado.

INEQUAÇÃO 2: A inequação 2 nos diz qual o peso limite que cada caminhão pode carregar. Como o C (Capacidade) é um valor fixo, todos os caminhões terão um mesmo limite de peso. Então cada item multiplicado pelo seu peso, terão que ser menores que a quantidade limite de peso que o caminhão consegue carregar. Essa restrição é importante, porque dessa forma conseguimos separar os itens em vários caminhões, caso contrário todos os itens irão em um único caminhão.

INEQUAÇÃO 3 e 4: Essas inequações apenas nos diz qual é o conjunto de valores que $x_{i,j} e y_j$ podem assumir.

2.1 FUNÇÃO DE CUSTO

Com a função abaixo que iremos conseguir calcular nosso resultado ótimo para o problema.

$$\sum_{j=1}^{n} y_j, 1 \leq j \leq n$$

Como o problema é encontrar a menor quantidade de viagens possíveis. Com isso, basta fazer a soma de todos os y_j que conseguiremos chegar em um resultado ótimo para o problema, é importante ressaltar que esse valor de y_j está dentro do limite de $\{0.0, 1.0\}$.

3 FORMATO DE ENTRADA

O nosso formato de entrada é a partir de um arquivo. Em sua primeira linha existem três números, **n** sendo a quantidade de itens, **p** indicando o número de pares ordenados, e **C** que indica a capacidade total de cada caminhão. Na segunda linha, tempos n pesos w_i, sendo que cada i representa o peso de um item. E no final, tempos os pares ordenados (p1, p2), que por motivos já citados anteriormente, não foram utilizados para resolver este problema. Para uma melhor representação, segue a imagem abaixo dê um exemplo genérico:

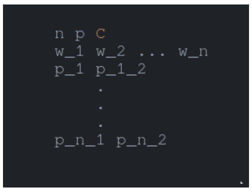

Imagem 6: Exemplo genérico de entrada.

Esse seria um exemplo genérico de entrada, para que o script consiga resolver independentemente dos valores de entrada.

3.1 EXEMPLO PRÁTICO

Neste capítulo vamos modelar o problema do envio de cargas de acordo com os valores de entrada da figura abaixo.

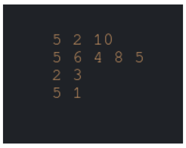

Imagem 7: Exemplo de entrada com valores reais.

Entrada:

n = 5; p = 2; C = 10;

$w_1 = 5, w_2 = 6, w_3 = 4, w_4 = 8, w_5 = 5$

$(p_1 = 2, p_2 = 3)$

$(p_1 = 5, p_2 = 1)$

Saída:

2.8

De acordo com o capítulo onde foi explicado as restrições e as variáveis, as restrições serão montadas de acordo com as inequações já apresentadas.

Inequação 1: $\sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} = 1$

As restrições ficariam da seguinte forma:

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = 1$$
$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = 1$$
$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = 1$$
$$x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} + x_{4,5} = 1$$

$x_{5,1} + x_{5,2} + x_{5,3} + x_{5,4} + x_{5,5} = 1$

Inequação 2: $\sum_{i=1}^{n} \sum_{j=1}^{n} w_i * x_{i,j} \leq C * y_i$

$5x_{1,1} + 5x_{1,2} + 5x_{1,3} + 5x_{1,4} + 5x_{1,5} \leq 10 * y_1$
$6x_{2,1} + 6x_{2,2} + 6x_{2,3} + 6x_{2,4} + 6x_{2,5} \leq 10 * y_2$
$4x_{3,1} + 4x_{3,2} + 4x_{3,3} + 4x_{3,4} + 4x_{3,5} \leq 10 * y_3$
$8x_{4,1} + 8x_{4,2} + 8x_{4,3} + 8x_{4,4} + 8x_{4,5} \leq 10 * y_4$
$5x_{5,1} + 5x_{5,2} + 5x_{5,3} + 5x_{5,4} + 5x_{5,5} \leq 10 * y_5$

limites:

$x_{i,j} \in \{0, 1\}$
$y_j \in \{0, 1\}$

Essa seria a modelagem do problema, se utilizarmos as restrições. Se jogarmos essas restrições em qualquer resolvedor, teríamos o resultado de 2.8. A imagem abaixo representa a solução dessa modelagem utilizando o lpsolve IDE, que é basicamente um resolvedor com interface gráfica.

Variables	result
	2.8
y1	0.5
y2	0.6
y3	0.4
y4	0.8
y5	0.5
x11	1
x12	0
x13	0
x14	0
x15	0
x21	1
x22	0
x23	0
x24	0
x25	0
x31	1
x32	0
x33	0
x34	0
x35	0
x41	1
x42	0
x43	0
x44	0
x45	0
x51	1
x52	0
x53	0
x54	0
x55	0

Imagem 8: Resolvedor com interface grafica IDE

4 MODELAGEM UTILIZANDO O PYTHON3

Neste capítulo será explicado como utilizar da ferramenta or-tools para resolver o problema de envio de cargas, será apresentado apenas as partes importantes do código. De forma que seja possível resolver o problema para qualquer entrada, separamos o código em duas partes. A primeira parte lê o arquivo de entrada e salvo em um dicionário chamado dictInput e a partir disso consigo criar um dicionário chamado data que representa a modelagem do problema de envio de cargas. E a na segunda parte é onde conseguimos a solução do problema.

```
# Dicionario que salvo os dados de entrada
# a partir do arquivo.
dictInput = {
    "firstLine": [],
    "weights": [],
    "ordered_pairs": [],
}
```

Imagem 9: Dicionário que contém o input dos dados.

```
# Indica os pesos de cada item
data["weights"] = dictInput["weights"]

# Indica a quantidade de itens existentes
# Será utilizado para criar as váriaveis mais tarde
data["items"] = list(range(len(dictInput["weights"])))

# Indica a quantidade de caminhões.
data["bins"] = data["items"]

# Indica os pares ordenados. Aviso: Não foi utilizado.
data["ordered_pairs"] = dictInput["ordered_pairs"]

#Indica a capacidade da carga, C
data["bin_capacity"] = dictInput["firstLine"][2]
```

Imagem 10: Dicionário que contém os dados modelados.

Agora que tempos os dados de entrada dentro de um dicionário, fica muito mais fácil de se criar as restrições do problema. Agora é necessário criar as variáveis dos itens e dos caminhões. Conseguimos fazer isso da seguinte forma:

```
# x[i, j] = 1, se o item está no caminhão j
x = {}
for i in data['items']:
    for j in data['bins']:
        # Cria uma váriavel(objeto do tipo NumVar) e salvo na váriavel x_i_j
        x[(i, j)] = solver.NumVar(0.0, 1.0, 'x_%i_%i' % (i, j))
```

Imagem 11: Criação das variáveis. $x_{i,j}$

Dessa forma, conseguimos criar as váriaveis que indicará em qual caminhão j o item i irá pertencer.

Para criarmos os possíveis caminhões, será feito algo similar.

```python
# y[j] = 1, se o caminhão j está sendo usado.
y = {}
for j in data['trucks']:
    y[j] = solver.NumVar(0, 1.0, 'y[%i]' % j)
```

Imagem 11: Criação das váriaveis dos caminhões y_j

Com isso conseguimos criar todas as variáveis necessárias para o problema.

Agora que as variáveis já estão criadas, é necessário criar as restrições do problema. Será criado primeiro a restrições que indicam em qual caminhão cada item será inserido, será feito isso da seguinte forma:

```python
# A soma dos itens existentes, não devem exceder 1.
# Isso porque, a soma dos itens não deve exceder o caminhão.
for i in data['items']:
    solver.Add(sum(x[i, j] for j in data['trucks']) == 1)
```

Imagem 12: Criação das restrições que indica o envio total do item;

Agora é preciso criar as restrições de pesos para cada caminão, será feito de forma similar as restrições anteriores.

```python
# Indica que o peso não deve exceder a capacidade
for j in data['trucks']:
    solver.Add(
        sum(x[(i, j)] * data['weights'][i] for i in data['items'])  <= data["trucks_capac
    )
```

Imagem 12: Criação das restrições de peso para cada caminhão.

Agora que todas as variáveis, restrições e toda a modelagem está feita, basta rodar o resolvedor fazendo o seguinte:

```python
# Função objetiva. Minimza o numero de caminhões utilizados.
solver.Minimize(solver.Sum([y[j] for j in data['trucks']]))

# Chama o resolver
solver.Solve()
```

Imagem 12: Função objetiva e resolvedor

Dessa forma é possível resolver qualquer situação que envolva o problema de envio de cargas citado neste artigo. Basta a entrada de dados serem no formato citado.

5 OUTROS EXEMPLOS

Entrada:

n = 10; p = 2; C = 20;

$w_1 = 5, w_2 = 6, w_3 = 4, w_4 = 8, w_5 = 5, w_6 = 0, w_7 = 1, w_8, = 2, w_9, = 39, w_{10} = 2$

$(p_1 = 2, p_2 = 3)$

$(p_1 = 5, p_2 = 1)$

Saída:

3.6

Entrada:

n = 15; p = 2; C = 30;

$w_1 = 7, w_2 = 2, w_3 = 4, w_4 = 4, w_5 = 5, w_6 = 29, w_7 = 2, w_8 = 1, w_9 = 94, w_{10} = 57, w_{11} = 12, w_{12} = 305, w_{13} = 28, w_{14} = 2, w_{15} = 3$

$(p_1 = 2, p_2 = 3)$

$(p_1 = 5, p_2 = 1)$

Saída:

Não existe solução ótima.

6 BRANCH AND BOUND

Até o capítulo anterior, a forma abordada para resolver o problema de envio ordenado foi utilizando do método simplex. A partir deste capítulo a abordagem utilizada para resolver o problema será outra, será utilizado o algoritmo o branch and bound para encontrar um resultado ótimo e esse ótimo deve ser um resultado inteiro.

6.1 RELEMBRANDO O PROBLEMA E METODOLOGIA UTILIZADA

Dado um conjunto de itens I com pesos W e pares ordenados (P1, P2), o objetivo é encontrar a melhor solução de forma que a quantidade de viagens realizadas que consiga transportar todos os itens sejam mínimas. Caso exista alguma combinação que satisfaça os requisitos da modelagem, a instância é considerada inviável. Isto é, o conjunto de itens devem respeitar as seguintes restrições:

- O conjunto I deve ter o mesmo tamanho de W. Se isso não for respeitado a instância é considerada inválida e o algoritmo não irá funcionar.
- As tuplas de pares ordenadas devem ser respeitadas. Caso isso não aconteça, a instância será considerada inválida.

Será utilizado o método de Branch and Bound para resolver o problema de envio ordenado, será utilizada apenas uma única função limitante.

6.2 VARIÁVEIS DO BRANCH AND BOUND

As variáveis utilizadas serão as mesmas das já apresentadas anteriormente.

Variáveis Gerais:

- n = Quantidade de itens que serão transportados.
- C = Capacidade total possível para cada caminhão.
- P = Quantidade de pares ordenados.

Váriaveis referente ao problema:

- w_i = representa o peso do item i.
- y_j = representa o caminhão j.
- $x_{i,j}$ = representa o item i no caminhão j.
- (p_1, p_2) = Representa uma tupla de pares ordenados.

Agora que as variáveis estão definidas, é possível rodar o algoritmo para resolver o problema.

7 IMPLEMENTAÇÃO DO CÓDIGO

Neste capítulo será exposto o funcionamento do programa, a entrada e a manipulação dos dados para desenvolver o Branch and Bound, a forma com que é feito a ramificação da árvore e como que a função limitante se comporta.

7.1 ENTRADA PADRÃO

Os dados são enviados através de um arquivo pela entrada padrão do sistema (stdin), onde na primeira linha contém o número de itens e caminhões (n), quantidade de pares ordenados (P) e a capacidade de cada caminhão (C). Na segunda linha contém n pesos ($0 \leq w_i \leq n$). E as linhas que sobrarem indicam os pares ordenados, sendo esses pares ordenados sempre em tuplas (p_0, p_1), pode-se ter mais de um par ordenado para um mesmo problema. Um exemplo de entrada seria o seguinte:

Entrada:

n = 5; P = 2; C = 10;
$w_1 = 5, w_2 = 6, w_3 = 4, w_4 = 8, w_5 = 5$
$(p_1 = 2, p_2 = 3)$
$(p_1 = 5, p_2 = 1)$

Após a leitura da entrada, os dados são colocados em um dicionário único, que envolve toda a modelagem do problema. Neste dicionário chamado "data"tempos 5 campos, "wheights"que indica os pesos na ordem de entrada, "items"que indica os itens em ordem crescente $0 \leq i \leq n$, "trucks"é a mesma ideia dos itens, tendo caminhões de $\leq i \leq n$, "ordered_pairs"que indica as tuplas de pares ordenados e por último tempos "trucks_capacity"que indica a capacidade máxima de cada caminhão, que é o mesmo valor para todos.

Ficando assim, para a entrada mostrado acima:

data["weights"] = [5, 6, 4, 8, 5]
data["items"] = [0, 1, 2, 3, 4]
data["trucks"] = [0, 1, 2, 3, 4]
data["ordered_pairs"] = [(2,3), (5, 1)]
data["trucks_capacity"] = 10

Como os dados de entrada já estão dentro do dicionário, é possível implementar o Branch and Bound.

7.2 RAMIFICAÇÃO E LIMITANTE

A raiz do B&B é o resultado da relaxação da modelagem mostrada nos capítulos anteriores. Se rodar o simplex para aquele primeiro resultado, é obtido um primeiro valor base. Esse valor base seria o resultado ótimo da raíz, o que pode ser qualquer valor, já que a raiz utiliza a modelagem relaxada. Com a raiz já calculada, será necessário ramificar a árvore. A estratégia de ramificação utilizada foi garantir que cada item n_i estivesse em um caminhão. A raiz possui o level 0 da árvore, os itens são atribuídos a cada caminhão de acordo com o simplex. Como cada item será atribuído para cada caminhão, quando a árvore for ramificada do level 0 para o 1, cada item n_i será atribuída ao caminhão 1. Ou seja, no level 1 terá um ramo onde o item 1 estará atribuído ao caminhão 1, terá um item 2 que será atribuído ao caminhão 1 até o último item ser atribuído ao caminhão 1. E isso vai se repetindo conforme a árvore for crescendo, quando a árvore for expandir para do level 1 para o level 2, teremos a garantia de que um item estará no caminhão 1 e o outro estará no caminhão 2, quando chegarmos a um level k, tem k itens atribuídos a k caminhões. Dessa forma teremos uma altura máxima da árvore de tamanho "n"que é a mesma da quantidade total de itens. Para cada level, a quantidade de itens disponíveis cai em 1. De forma que no level n, terá apenas um único item sobrando para ser atribuído. Essa foi a estratégia utilizada para fazer a ramificação de todos os itens na árvore, garantindo que a solução fosse encontrada em uma árvore de altura n.

Porém, é inviável e desnecessário procurar uma solução em uma árvore dessas. Para evitar todo esse espaço de busca, foram utilizadas algumas estratégias para poder podar a árvore. A primeira foi garantir que os pares ordenados sejam respeitados. Pegando o exemplo da seção anterior, o primeiro par (2,3). Como o item 2 deve ser estar em uma viagem antes do item 3, na ramificação da árvore, todos os ramos em que o 3 foi atribuído a um caminhão antes do 2, o ramo será podado diretamente. Porque isso deixa a modelagem do problema inviável. E isso também é válido para qualquer outro par ordenado. Outra estratégia utilizada foi a poda por inviabilidade. Se após as atribuições de itens a caminhões em um determinado nodo da árvore, transformar o problema inviável, teremos uma poda naquele ramo da árvore.

E a última estratégia utilizada foi a poda por limitante. O limitante utilizado para o B&B é a própria função objetiva da modelagem:

$$\sum_{j=1}^{n} y_j, 1 \leq j \leq n$$

Com essa função, é possível garantir sempre a menor quantidade de viagens possíveis. Essas foram as estratégias de ramificações e podas utilizadas.

7.3 EXEMPLO PRÁTICO

Para ficar mais fácil a explicação das coisas acontecendo, será utilizado a seguinte árvore com a seguinte entrada.

Entrada:

n = 3; P = 1; C = 10;

$w_1 = 5, w_2 = 6, w_3 = 4$

$(p_1 = 1, p_2 = 2)$

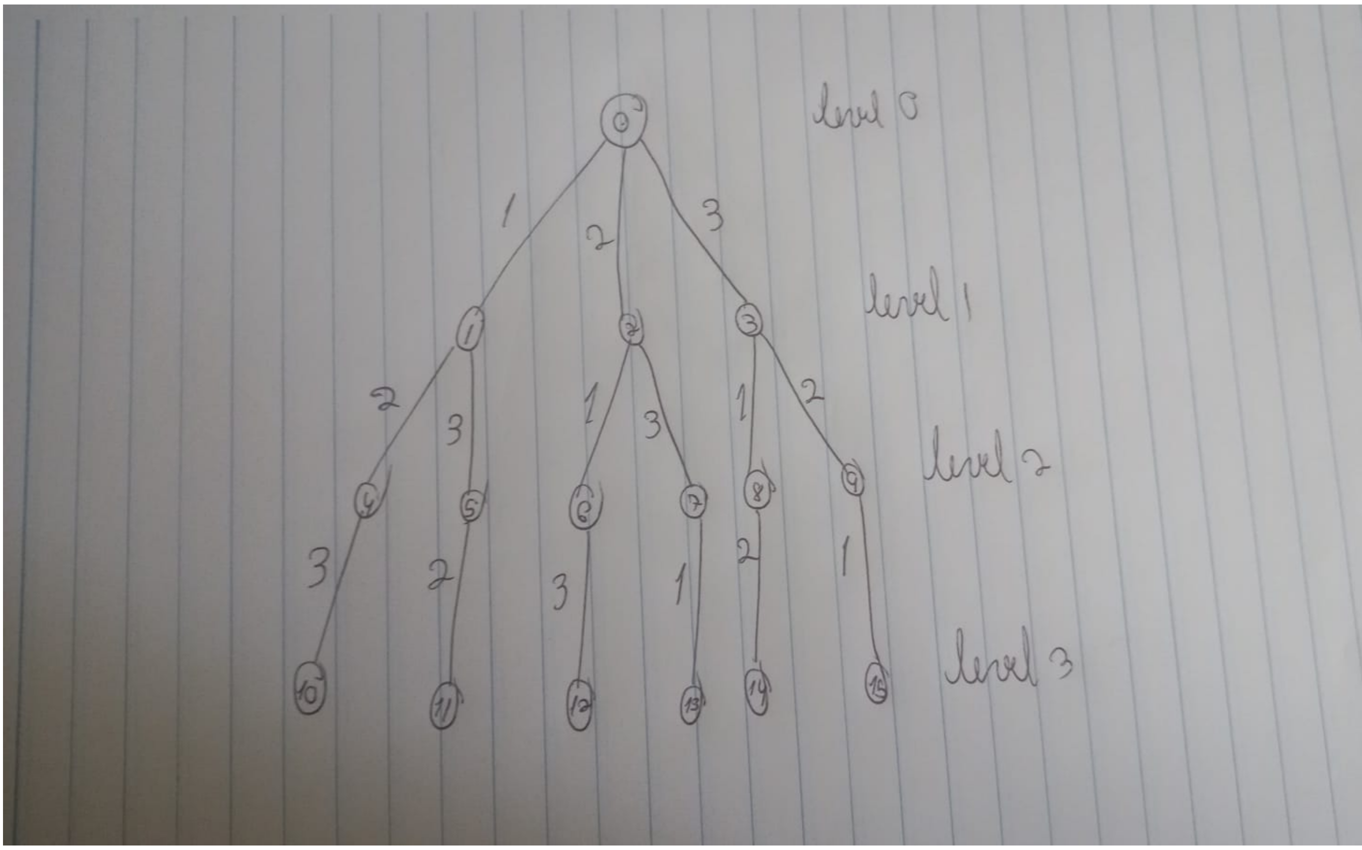

Imagem 13: Exemplo da árvore ramificada e sem podas.

A raiz possui um id de 0. A ramificação acontece da esquerda para a direita, então os nodos mais à esquerda irão possuir um id menor do que os nodos mais a direita. Como explicado Na seção anterior, o level 0 devolve uma solução relaxada do problema e começa a ramificar da esquerda para a direita. Neste exemplo, teremos 3 possíveis ramificações, aloca o item 1 para o caminhão 1 no primeiro ramo, aloca o item 2 no caminhão 1 no segundo ramo e aloca o item 3 no caminhão 1 no terceiro ramo. Porém devido aos pares ordenados, o item 2 não será mais expandido além do level 1. Isso porque neste ramo o item 2 é alocado antes do item 1, o que transforma a modelagem inviável. Dessa forma, os possíveis ramos que podem ser explorados até o momento são os seguintes.

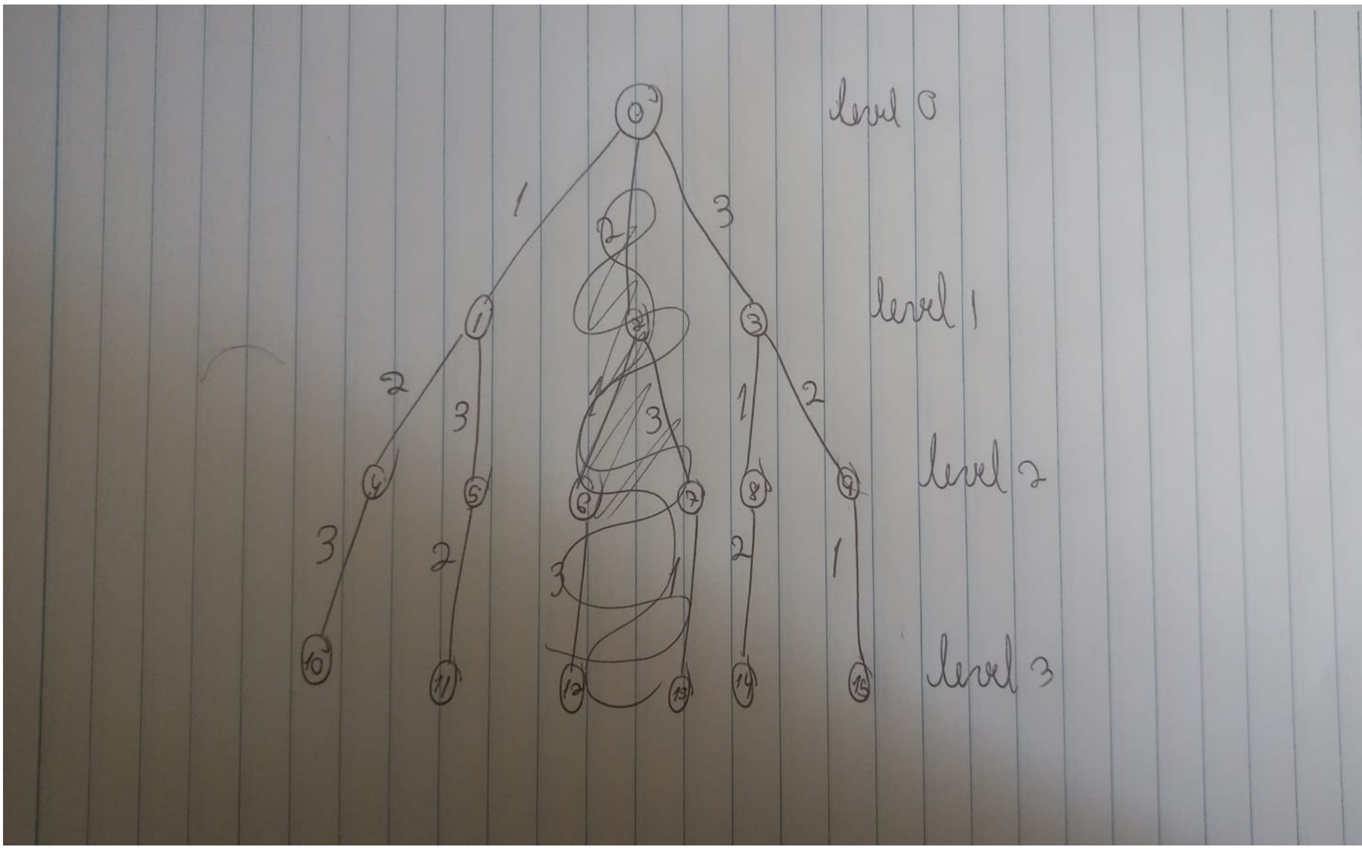

Imagem 13: Exemplo da árvore ramificada e sem podas.

Após o level 1, as modificações aconteceram do level 1 para o level 2. No primeiro ramo mais à esquerda, teremos o item 1 no caminhão 1, e o item 2 e 3 em ramos separados nos caminhões 2. E o mesmo acontece para o ramo mais à direita. Porém, se colocar o item 3 no caminhão 1 e o item 2 no caminhão 2 temos uma poda. Isso porque, acontecerá inviabilidade nos pares ordenados, porque o item 2 não pode vir antes do item 1. Então é possível podar aquele ramo também. A árvore ficaria da seguinte forma.

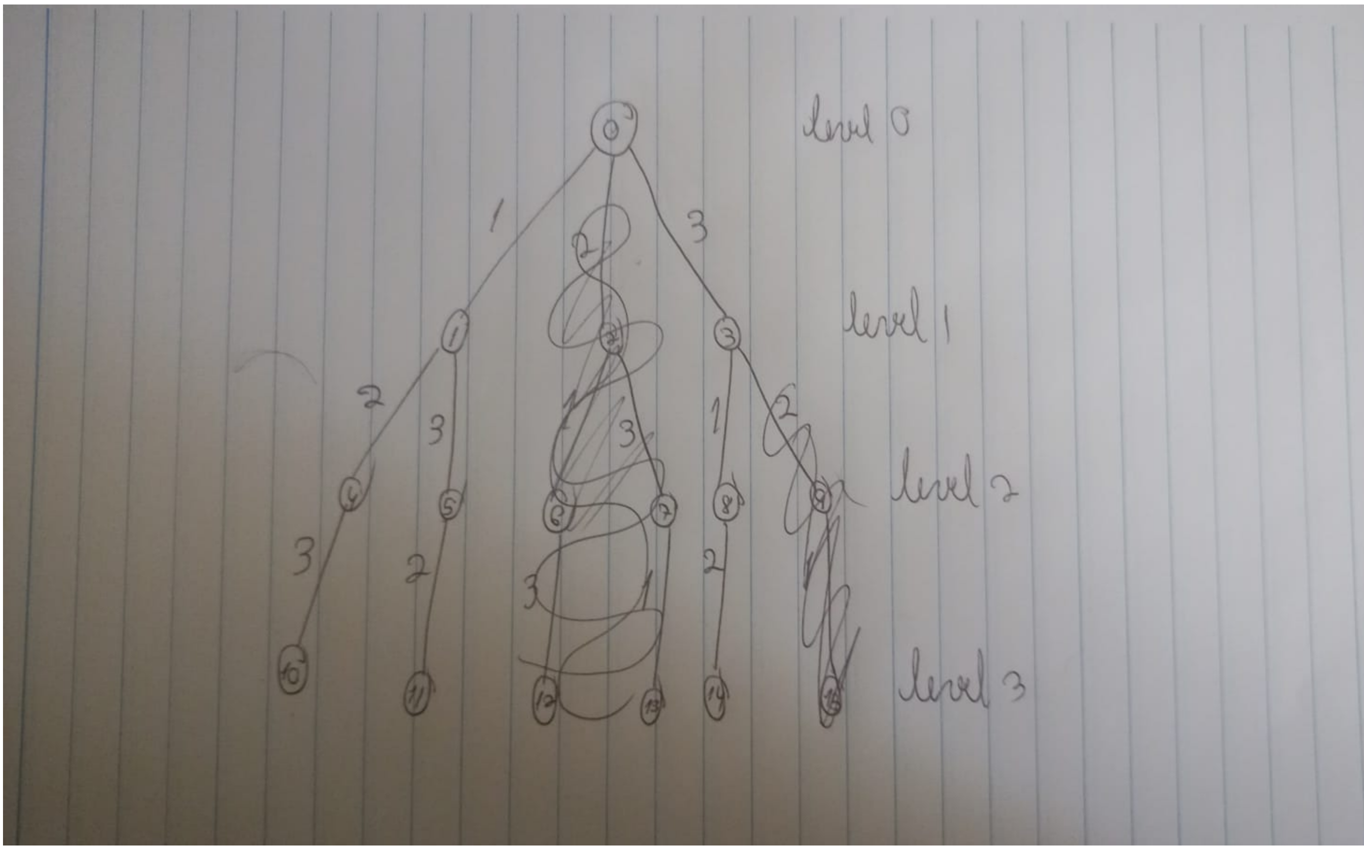

Imagem 13: Exemplo da árvore ramificada e sem podas.

E do level 2 para o 3, não terá mais nenhum problema com as ordenações. E como a função objetiva devolverá sempre o mesmo valor, não terá nenhuma poda por causa da função limitante.

8 EXPERIMENTO E RESULTADOS

n	Pres Ord.	C	Nodos Explorados	Resultado
5	(2,3), (5,1)	10	108	2.8 (Raiz)
10	(2,3), (5,1), (6, 7), (1, 2)	20	51434	3.6 (Raiz)

**Nota: Os testes estão em ./testes

Quanto maior a entrada maior a quantidade de nodos explorados. Isso acontece porque não existe poda por limitante, por causa do limitante não ser bom. Podendo chegar a ser uma quantidade extremamente alta de nodos explorados.

9 CONCLUSÃO

A função limitante encontrada não é uma função boa. Isso acontece porque, ela não poda nenhum elemento da árvore, porque todos os resultados ótimos de cada nodo, retornará o mesmo valor. Então as podas acontecem apenas para os ramos em que os pares ordenados não são respeitados.

10 REFERÊNCIAS

[1] MATOUŠEK, Jiří; GÄRTNER, Bernd. Understanding and Using Linear Programming. [S. l.]: Springer, 2007.

[2] Google OR-Tools, Google, https://developers.google.com/optimization, 2015, acessado em 11/04/2022