Investigação Operacional – Trabalho Prático I

Problema de Empacotamento a Uma Dimensão

Ana Carolina Penha Cerqueira	A104188
Humberto Gil Azevedo Sampaio Gomes	A104348
Ivo Filipe Mendes Vieira	A103999
José António Fernandes Alves Lopes	A104541
José Rodrigo Ferreira Matos	A100612

23 de março de 2024

Resumo

Este trabalho prático de Investigação Operacional tem como objetivo a resolução de um problema de empacotamento a uma dimensão utilizando o modelo de "um-corte", de Dyckhoff. [1] Em detalhe, procura-se a formulação do problema, a sua modelação, a sua resolução, e a validação do modelo construído. Para tornar o processo de modelação mais rápido e menos propício a erros, o nosso grupo implementou conceitos de metaprogramação ao escrever um *script* Python que automaticamente gera o código LP do modelo.

1 Dados do problema

Como exigido pelo enunciado, os dados do problema a resolver são determinados em função do número de aluno mais elevado de entre os integrantes do grupo. No caso, este é A104541, que corresponde ao aluno José Lopes, e dá origem aos seguintes dados:

Capacidade	Quantidade de contentores
11	Ilimitada
10	5
7	5

Tabela 1: Número de contentores de cada comprimento disponíveis.

Comprimento	Quantidade de itens
1	0
2	13
3	0
4	9
5	5

Tabela 2: Número de itens de cada comprimento para empacotar.

A soma dos comprimentos dos itens a empacotar é dada por:

$$2 \times 13 + 4 \times 9 + 5 \times 5 = 87 \tag{1}$$

2 Formulação do problema

Pretende-se resolver um problema de empacotamento a uma dimensão, i.e., pretende-se determinar como se deve distribuir um conjunto de itens por contentores. É necessário ter em conta que deve ser atribuído exatamente um contentor a cada item, ou seja, um dado item não pode estar em mais do que um contentor, e não pode haver itens que não tenham um contentor associado. Ademais, a soma dos comprimentos dos itens em qualquer contentor não pode ultrapassar a sua capacidade.

Neste problema em específico, estão disponíveis contentores de capacidades 11, 10 e 7, sendo que há contentores de capacidade 11 em número ilimitado, enquanto que há apenas 5 contentores de cada uma das outras capacidades (tabela 1). Pretendem-se empacotar 13 itens de comprimento 2, 9 itens de comprimento 4, e 5 itens de comprimento 5 (tabela 2). O objetivo é minimizar a soma das capacidade dos contentores utilizados.

2.1 Modelo de Dyckhoff

O modelo de "um-corte", de Dyckhoff [1], assenta na ideia do "padrão-um-corte", uma divisão de um comprimento k em dois menores, l e $k-l$, representada pelo tuplo $[k; l]$. Note-se que nesta secção adaptaremos o vocabulário do modelo, referente a um problema de *cutting stock*, para o nosso problema de *bin packing*. Estes cortes são aplicados recursivamente, começando com valores de k iguais às capacidades dos contentores, e valores de l sempre iguais aos comprimentos dos itens. Os resíduos provenientes do corte $[k; l]$ podem depois ser divididos em partes menores, e assim sucessivamente.

São denominados resíduos úteis todos os comprimentos que não são inferiores ao mais curto dos itens, e que podem ser obtidos através de "um-cortes" sucessivos a partir das capacidades dos contentores e dos comprimentos dos itens. Da perspetiva do modelo de Dyckhoff, dois resíduos do mesmo comprimento são equivalentes, sejam eles capacidades de contentores ou resultados de "um-cortes" anteriores.

Neste modelo, cada variável de decisão $y_{k,l}$ corresponde ao número de comprimentos k divididos de acordo com o "um-corte" $[k; l]$. Com base numa solução de um problema por este modelo, é possível associar os "um-cortes" à distribuição dos itens pelos contentores, apesar de poder haver várias interpretações possíveis para o mesmo conjunto de valores das variáveis.

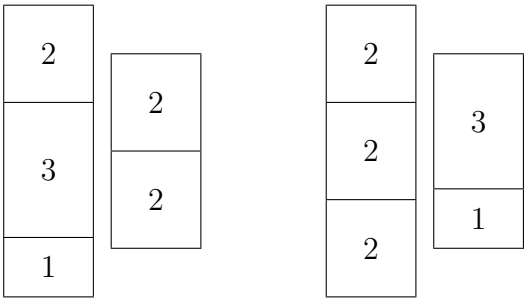

Figura 1: Duas possíveis interpretações para o mesmo conjunto de valores de variáveis de decisão: $y_{6,2} = 1$, $y_{4,2} = 1$ e $y_{4,3} = 1$.

A partir das variáveis de decisão podemos construir o único tipo de restrição que existe no modelo original de Dyckhoff, denominada de equilíbrio. As restrições deste tipo têm como

objetivo assegurar dois aspetos importantes do modelo. Primeiramente, é exigido que, no fim da aplicação dos "um-cortes", se tenham produzido espaços suficientes para empacotar todos os itens dentro dos contentores, tendo cada item um espaço reservado de comprimento igual a si mesmo. Além disso, para resíduos que não sejam capacidades de contentores, não podem ser cortados mais comprimentos do que aqueles que foram produzidos a partir da aplicação de "um-cortes". Sendo assim, matematicamente, a diferença entre os comprimentos produzidos e consumidos através de "um-cortes" deve ser superior a um valor mínimo, o número de itens de um comprimento a empacotar no primeiro caso, ou zero no segundo.

O objetivo do modelo de "um-corte" é o de minimizar o custo dos contentores utilizados. No nosso caso, o custo de um contentor é a sua capacidade. Logo, a função objetivo é dada pela soma, para cada tipo de contentor, do produto entre a sua capacidade e o seu número de usos. Trivialmente, o número de usos de cada tipo de contentor é dado pela diferença entre o número de "um-cortes" executados que consomem o seu comprimento, e o número de "um-cortes" que o produzem.

2.2 Extensões ao modelo de Dyckhoff

Existem variáveis de decisão que têm o mesmo significado físico ($y_{k,l}$ e $y_{k,k-l}$), como por exemplo $y_{10,3}$ e $y_{10,7}$. Enquanto que o modelo original não descarta nenhuma variável, descartar uma das anteriores é possível, e foi como prosseguimos neste trabalho.

Ademais, o modelo de "um-corte" não estabelece restrições para a disponibilidade de cada tipo de contentor, algo necessário para o nosso problema. O cálculo do número de contentores de uma dada capacidade que foram usados, em função dos "um-cortes" executados, já foi descrito na construção da função objetivo. Basta garantir que este número não é superior à disponibilidade do contentor.

Por último, o modelo de Dyckhoff original considera que todos os resíduos são iguais. Por exemplo, num problema de corte onde se aplique este modelo, é possível que sobrem resíduos com comprimentos iguais aos do *stock* a cortar. O modelo considera que o número de rolos gastos destes comprimentos é negativo, e o valor da função objetivo (custo) diminui, considerando que os rolos produzidos podem ser guardados para uso posterior. No entanto, isto não faz qualquer sentido num problema de empacotamento, onde sobras de espaço em contentores não devem ser contabilizadas para a função objetivo: para cada capacidade de contentor, considera-se o valor máximo entre zero e o custo dos contentores usados.

3 Modelo de "um-corte" para um problema genérico

3.1 Parâmetros e notação utilizada

Antes de apresentar o modelo final adaptado ao nosso problema, apresentaremos a formulação matemática do modelo de "um-corte" para um caso geral, complementando a descrição meramente textual feita anteriormente. Alguns objetos matemáticos devem ser conhecidos antes da introdução da função objetivo e das restrições existentes. Será seguida a mesma notação do artigo que definiu este modelo. [1]

- $S \subset \mathbb{Z}^+$: conjunto dos comprimentos de *stock*, no caso, as capacidades dos contentores;
- $D \subset \mathbb{Z}^+$: conjunto dos comprimentos do pedido, no caso, os comprimentos dos itens;
- $R \subset \mathbb{Z}^+$: conjunto de resíduos relevantes, definidos como comprimentos não inferiores ao menor dos itens, que podem ser obtidos através da aplicação de "um-cortes" sucessivos a partir de S e D.

Neste modelo, são também dados do problema os seguintes parâmetros:

- $N_l \in \mathbb{Z}_0^+$, com $l \in (D \cup R) \setminus S$: número de itens a empacotar com o comprimento l (quando $l \notin D$, $N_l = 0$);
- $c_l \in \mathbb{R}$, com $l \in S$: custo de um contentor com comprimento l;
- $d_l \in \mathbb{Z}^+ \cup \{\infty\}$, com $l \in S$: disponibilidade de um contentor de uma dada capacidade.

3.2 Variáveis de decisão

Como já explicado, as variáveis de decisão principais deste problema são da forma $y_{k,l}$, e o seu valor representa o número de divisões de comprimentos k em comprimentos menores, l e $k - l$. Algumas regras triviais aplicam-se aos valores de k e l:

- O número de cortes é inteiro e não negativo: $y_{k,l} \in \mathbb{Z}_0^+$;
- De um "um-corte" não podem resultar comprimentos maiores ou iguais do que o comprimento cortado: $l < k$;

- De qualquer "um-corte", um dos comprimentos produzidos deve ser o comprimento de um item: $l \in D$;
- Apenas se podem dividir capacidades de contentores ou de comprimentos obtidos através de "um-cortes" anteriores (resíduos): $k \in S \cup R$.

O conjunto de variáveis no modelo final não é conhecido antes da expansão da função objetivo e das restrições.

3.3 Função objetivo

Na nossa extensão ao modelo de Dyckhoff, são necessárias outras variáveis de decisão para a linearização da função objetivo. Procura-se minimizar a soma dos custos dos contentores utilizados, assegurando-se que o número de usos de um dado contentor não é negativo:

$$\text{min: } \sum_{l \in S} c_l \cdot \max \left\{ 0, \sum_{k \in C_l} y_{l,k} - \sum_{k \in B_l} y_{k+l,k} \right\} \tag{2}$$

Na equação anterior, os conjuntos B_l e C_l assumem os seguintes valores:

$$B_l = \{k \in D \mid k + l \in S \cup R\}$$

$$C_l = \{k \in D \mid k < l\}$$

Tendo em conta as definições de B_l e C_l, conclui-se que o somatório a azul na função objetivo (2) representa o número de comprimentos l consumidos, enquanto que o somatório a verde representa o número de comprimentos l produzidos através de "um-cortes".

A linearização desta função objetivo é um processo simples: para cada $l \in S$, considera-se uma variável de decisão m_l, que representa o máximo entre 0 e a diferença entre os somatórios. Sendo assim, m_l será superior ou igual a ambos estes valores, e a minimização da soma de todos os m_l conduzirá a que cada uma destas variáveis assuma exatamente um dos valores (0 ou a diferença entre somatórios), visto que a solução ótima estará na fronteira de um poliedro convexo:

$$\begin{aligned} &\text{min: } \sum_{l \in S} c_l \cdot m_l \\ &\forall_{l \in S} \begin{cases} m_l \geq 0, \\ m_l \geq \sum_{k \in C_l} y_{l,k} - \sum_{k \in B_l} y_{k+l,k} \end{cases} \end{aligned} \tag{3}$$

3.4 Restrições

Ao contrário das restrições anteriores, existentes apenas para ser possível uma representação linear do modelo, segue-se a apresentação das restrições impostas pelo contexto e dados do problema.

3.4.1 Restrições de equilíbrio

Estas restrições são definidas no artigo de Dyckhoff [1] e garantem dois aspetos: produz-se o número de espaços suficientes para empacotar os itens necessários, e não se consomem mais comprimentos em "um-cortes" do que os que são produzidos. Para cada comprimento desejado, ou resíduo que não igual à capacidade de um contentor ($l \in (D \cup R) \setminus S$), tem-se a seguinte restrição:

$$\sum_{k \in A_l} y_{k,l} + \sum_{k \in B_l} y_{k+l,k} - \sum_{k \in C_l} y_{l,k} \geq N_l \tag{4}$$

Os conjuntos B_l e C_l já foram definidos anteriormente, e a definição de A_l segue-se abaixo:

$$A_l = \begin{cases} \{k \in S \cup R \mid k > l\}, & l \in D \\ \varnothing, & l \notin D \end{cases}$$

Assim, os somatórios a vermelho e verde representam o número de espaços gerados com comprimento l, enquanto que o somatório a azul representa o número desses espaços consumidos por "um-cortes" subsequentes. O balanço final de cortes não deve ser inferior a 0, e nos casos em que há itens a empacotar, N_l.

3.4.2 Restrições de contentores

Devido à disponibilidade limitada de cada contentor, é necessário estabelecer restrições que garantam que o número de contentores usado de cada tipo (já calculado na função objetivo) não ultrapasse a sua disponibilidade. Para cada $l : d_l \neq \infty$, tem-se a seguinte restrição:

$$\sum_{k \in C_l} y_{l,k} - \sum_{k \in B_l} y_{k+l,k} \leq d_l \tag{5}$$

Apesar de denominarmos estas restrições "de contentores", tal foi feito apenas para as distinguir das do modelo original de Dyckhoff. No entanto, podemos provar que estas se tratam de meras restrições de equilíbrio que consideram $N_l = -d_l$ (não se pode ultrapassar um dado consumo de contentores):

$$\begin{aligned} \sum_{k \in C_l} y_{l,k} - \sum_{k \in B_l} y_{k+l,k} \leq d_l \Leftrightarrow \\ 0 + \sum_{k \in B_l} y_{k+l,k} - \sum_{k \in C_l} y_{l,k} \geq -d_l \Leftrightarrow_{A=\varnothing} \\ \sum_{k \in A_l} y_{k,l} + \sum_{k \in B_l} y_{k+l,k} - \sum_{k \in C_l} y_{l,k} \geq -d_l = N_l \end{aligned}$$

4 Aplicação do modelo de "um-corte" ao nosso problema

Para os parâmetros do nosso problema, consideramos $S = \{7, 10, 11\}$, as capacidades dos contentores, $D = \{2, 4, 5\}$, os comprimentos dos itens, e $R = \{2, 3, 4, 5, 6, 7, 8, 9\}$, os comprimentos dos resíduos úteis, calculados com base na aplicação de "um-cortes" sucessivos:

$$\begin{aligned} R_0 &\leftarrow \varnothing \\ [11; 2] : R_1 &\leftarrow R_0 \cup \{2, 9\} \\ [11; 4] : R_2 &\leftarrow R_1 \cup \{4, 7\} \\ [11; 5] : R_3 &\leftarrow R_2 \cup \{5, 6\} \\ [10; 2] : R_4 &\leftarrow R_3 \cup \{2, 8\} \\ &\vdots \end{aligned}$$

Dado o número de itens de cada tipo a empacotar, $N_2 = 13$, $N_4 = 9$ e $N_5 = 5$. Ademais, com a disponibilidade de contentores de cada tipo, $d_7 = d_{10} = 5$ e $d_{11} = \infty$. Sabemos ainda que cada contentor tem um custo igual à sua capacidade, dado que procuramos minimizar a soma dos comprimentos dos contentores utilizados: $\forall_{l \in S}, c_l = l$.

Antes da expansão das restrições, não sabemos quais serão as variáveis de decisão que estarão presentes no modelo final. No entanto, tendo já feito isso, tem-se:

- Variáveis necessárias para a linearização da função objetivo (m_l representa o número de contentores usados de capacidade l): m_7, m_{10}, $m_{11} \in \mathbb{Z}_0^+$
- Variáveis de corte (com $y_{k,l}$ a representar o número de comprimentos k divididos em l e $k - l$): $y_{11,5}$, $y_{11,4}$, $y_{11,2}$, $y_{10,5}$, $y_{10,4}$, $y_{10,2}$, $y_{9,2}$, $y_{7,5}$, $y_{7,4}$, $y_{8,2}$, $y_{6,4}$, $y_{5,2}$, $y_{4,2}$, $y_{3,2}$, $y_{8,5}$, $y_{9,5}$, $y_{8,4}$, $y_{5,4}$, $y_{6,5} \in \mathbb{Z}_0^+$

A expansão da expressão em (3) resulta na seguinte função objetivo e restrições:

$$\begin{aligned} \text{min: } & 11m_{11} + 10m_{10} + 7m_7 \\ \text{Suj. a: } & m_{11} \geq y_{11,5} + y_{11,4} + y_{11,2} \\ & m_{10} \geq y_{10,5} + y_{10,4} + y_{10,2} \\ & m_7 \geq -y_{11,4} - y_{9,2} + y_{7,5} + y_{7,4} \end{aligned}$$

A expansão das restrições de equilíbrio em (4) resulta nas seguintes expressões:

$$\begin{aligned} y_{11,2} + y_{10,2} + y_{9,2} + y_{8,2} + y_{7,5} + y_{6,4} + y_{5,2} + 2y_{4,2} + y_{3,2} &\geq 13 \\ y_{8,5} + y_{7,4} + y_{5,2} - y_{3,2} &\geq 0 \\ y_{11,4} + y_{10,4} + y_{9,5} + 2y_{8,4} + y_{7,4} + y_{6,4} + y_{5,4} - y_{4,2} &\geq 9 \\ y_{11,5} + 2y_{10,5} + y_{9,5} + y_{8,5} + y_{7,5} + y_{6,5} - y_{5,4} - y_{5,2} &\geq 5 \\ y_{11,5} + y_{10,4} + y_{8,2} - y_{6,5} - y_{6,4} &\geq 0 \\ y_{10,2} - y_{8,5} - y_{8,4} - y_{8,2} &\geq 0 \\ y_{11,2} - y_{9,5} - y_{9,2} &\geq 0 \end{aligned}$$

Ademais, as restrições relativas ao número máximo de contentores (5) são expandidas para o seguinte:

$$\begin{aligned} y_{10,5} + y_{10,4} + y_{10,2} &\leq 5 \\ -y_{11,4} - y_{9,2} + y_{7,5} + y_{7,4} &\leq 5 \end{aligned} \tag{6}$$

É de notar que omitimos as restrições do tipo $m_l, y_{k,l} \geq 0$, pois estas são asseguradas pelo tipo das variáveis (inteiros não negativos), tanto matematicamente como no `lpsolve`. Segue-se o nosso modelo completo:

$$\begin{aligned} \text{min: } & 11m_{11} + 10m_{10} + 7m_7 \\ \text{Suj. a: } & m_{11} \geq y_{11,5} + y_{11,4} + y_{11,2} \\ & m_{10} \geq y_{10,5} + y_{10,4} + y_{10,2} \\ & m_7 \geq -y_{11,4} - y_{9,2} + y_{7,5} + y_{7,4} \\ \\ & y_{11,2} + y_{10,2} + y_{9,2} + y_{8,2} + y_{7,5} + y_{6,4} + y_{5,2} + 2y_{4,2} + y_{3,2} \geq 13 \\ & y_{8,5} + y_{7,4} + y_{5,2} - y_{3,2} \geq 0 \\ & y_{11,4} + y_{10,4} + y_{9,5} + 2y_{8,4} + y_{7,4} + y_{6,4} + y_{5,4} - y_{4,2} \geq 9 \\ & y_{11,5} + 2y_{10,5} + y_{9,5} + y_{8,5} + y_{7,5} + y_{6,5} - y_{5,4} - y_{5,2} \geq 5 \\ & y_{11,5} + y_{10,4} + y_{8,2} - y_{6,5} - y_{6,4} \geq 0 \\ & y_{10,2} - y_{8,5} - y_{8,4} - y_{8,2} \geq 0 \\ & y_{11,2} - y_{9,5} - y_{9,2} \geq 0 \\ \\ & y_{10,5} + y_{10,4} + y_{10,2} \leq 5 \\ & -y_{11,4} - y_{9,2} + y_{7,5} + y_{7,4} \leq 5 \\ \\ & y_{l,k}, m_l \in \mathbb{Z}_0^+ \end{aligned}$$

5 Ficheiro de entrada do lpsolve

```
1  min: 11 m11 + 10 m10 + 7 m7;
2
3  /* Standardize optimizations of max{0, ...} */
4  m11 >= y11_5 + y11_4 + y11_2;
5  m10 >= y10_5 + y10_4 + y10_2;
6  m7 >= -1 y11_4 + -1 y9_2 + y7_5 + y7_4;
7
```

```
/* Balance restrictions */
l2: y11_2 + y10_2 + y9_2 + y8_2 + y7_5 + y6_4 + y5_2 + 2 y4_2 + y3_2 >=
    13;
l3: y8_5 + y7_4 + y5_2 + -1 y3_2 >= 0;
l4: y11_4 + y10_4 + y9_5 + 2 y8_4 + y7_4 + y6_4 + y5_4 + -1 y4_2 >= 9;
l5: y11_5 + 2 y10_5 + y9_5 + y8_5 + y7_5 + y6_5 + -1 y5_4 + -1 y5_2 >=
   5;
l6: y11_5 + y10_4 + y8_2 + -1 y6_5 + -1 y6_4 >= 0;
l8: y10_2 + -1 y8_5 + -1 y8_4 + -1 y8_2 >= 0;
l9: y11_2 + -1 y9_5 + -1 y9_2 >= 0;

/* Container upper bounds */
c10: y10_5 + y10_4 + y10_2 <= 5;
c7: -1 y11_4 + -1 y9_2 + y7_5 + y7_4 <= 5;

int y10_2, y10_4, y10_5, y11_2, y11_4, y11_5, y3_2, y4_2, y5_2, y5_4,
   y6_4, y6_5, y7_4, y7_5, y8_2, y8_4, y8_5, y9_2, y9_5;
```

6 Ficheiro de saída do lpsolve

```
Value of objective function: 87.00000000

Actual values of the variables:
m11                             3
m10                             4
m7                              2
y11_5                           0
y11_4                           0
y11_2                           3
y10_5                           0
y10_4                           4
y10_2                           0
y9_2                            0
y7_5                            2
y7_4                            0
y8_2                            0
y6_4                            4
y5_2                            0
y4_2                            2
y3_2                            0
y8_5                            0
y9_5                            3
y8_4                            0
y5_4                            0
y6_5                            0
```

7 Interpretação da solução ótima

Os valores das variáveis de decisão apresentados na secção acima podem ser associados a uma distribuição dos itens por contentores. Por observação dos valores das variáveis m_l, concluímos sobre o número de contentores de cada tipo utilizados:

Capacidade	Contentores utilizados
11	3
10	4
7	2

Tabela 3: Número de contentores de cada tipo utilizados na solução ótima do nosso problema.

Com base nas variáveis de corte ($y_{k,l}$), podemos agora ir dividindo os comprimentos dos contentores, executando o número de "um-cortes" de cada tipo especificado pela solução. A distribuição de itens por contentores que deduzimos é a seguinte:

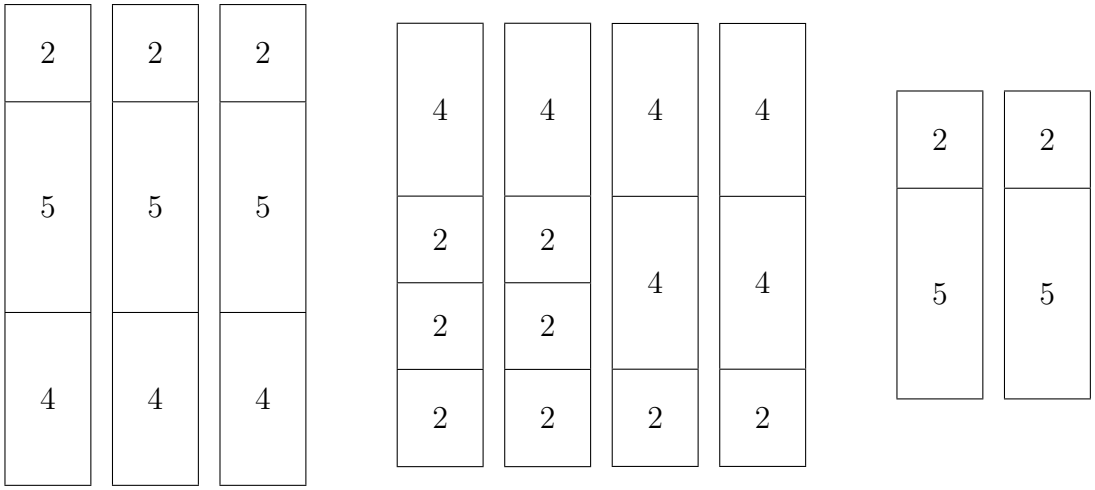

Figura 2: Uma distribuição possível dos itens pelos contentores na solução ótima.

O valor da função objetivo, correspondente à soma dos comprimentos dos contentores utilizados, é 87.

8 Validação do modelo

Após o cálculo da solução ótima, procedemos à validação do modelo. O primeiro passo consiste em verificar se a interpretação real da solução obtida faz sentido, e não viola nenhum pressuposto

que possa ter sido esquecido no modelo LP. Verifica-se que:

- Todos os itens foram empacotados: 13 de comprimento 12, 9 de comprimento 4 e 5 de comprimento 5;
- Não foi ultrapassada a capacidade de nenhum dos contentores;
- Não foram usados mais contentores do que os disponíveis: apenas 3 de capacidade 10, e 2 de capacidade 7, números inferiores aos 5 contentores disponíveis para cada um destes tipos.

De seguida, verificamos se o nosso modelo não é demasiado restritivo, i.e., se não existem soluções melhores do que a atual, válidas no mundo real mas que não respeitam as restrições do modelo LP. Verificamos que a soma dos comprimentos dos itens (ver equação 1) é igual à soma das capacidades dos contentores utilizados, 87, pelo que a nossa solução não gera qualquer espaço livre, e consequentemente não existe melhor solução possível que não seja considerada pelo nosso modelo.

9 Metaprogramação

A expansão dos vários somatórios do modelo de "um-corte" é, devido à sua dimensão, difícil de ser executada manualmente. O processo não só é demorado, como também propício a erros de cálculo difíceis de diagnosticar: uma solução errada ou a ausência de solução no modelo final apenas indica a existência de um erro, mas não a sua localização. Ademais, o processo laborioso de criação do modelo teria de ser repetido caso os dados iniciais do problema sofressem alterações, uma ocorrência constante no mundo real. Para endereçar este problema, desenvolvemos *scripts* em Python que geram o modelo LP automaticamente.

O nosso *script* para a implementação deste modelo encontra-se em anexo (ver 12.1) e o seu funcionamento passo a passo é descrito abaixo.

1. Calcular o conjunto de resíduos úteis (R), sucessivamente executando todos os cortes possíveis com base nas capacidades dos contentores e nos comprimentos dos itens.
2. Gerar a função objetivo, em particular, a variação que não diminui o custo de uma solução caso tenham sobrado espaços com comprimentos no conjunto das capacidades dos contentores (ver equação 3).

3. Gerar as restrições de equilíbrio, calculando os conjuntos e os somatórios descritos em (4).

4. Gerar restrições a afirmar que o número de contentores de um dado tipo utilizados não pode ser superior à disponibilidade dessa capacidade de contentor (ver equação 5).

5. Restringir todas as variáveis de corte a valores inteiros.

No nosso *script*, a remoção de variáveis redundantes é feita no processo de simplificação de adições, tanto na geração da função objetivo como das restrições: $(y_{10,2}) + (y_{10,8} + y_{8,4})$ será automaticamente transformada em $y_{10,2} + y_{8,4}$, por exemplo.

10 Conclusão

Ao longo do desenvolvimento deste trabalho, não só fomos capazes de resolver o problema proposto com correção, como também obtivemos um profundo conhecimento de um modelo para a resolução de problemas de empacotamento a uma dimensão. Este conhecimento permitiu-nos implementar um programa que constrói este modelo sem qualquer intervenção humana, tornando a sua aplicação mais simples em problemas maiores e em contextos em que os dados estão em constante mudança, evitando o desperdício de horas humanas em cálculos que facilmente podem ser executados por um computador.

11 Bibliografia

[1] H. Dyckhoff, "A New Linear Programming Approach to the Cutting Stock Problem", *Operations Research*, vol. 29, no. 6, Dec., pp. 1092-1104, 1981. doi: 10.1287/opre.29.6.1092

12 Anexos

12.1 *Script* gerador do modelo de "um-corte"

```python
#!/usr/bin/env python3

# This program is free software: you can redistribute it and/or modify it under the terms of the GNU
# General Public License as published by the Free Software Foundation, either version 3 of the
# License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
# even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with this program. If not,
# see <https://www.gnu.org/licenses/>.

from collections.abc import Iterable
from typing import Optional

CutVar = tuple[float, float]
""" A variable that represents a one-cut [l;k]. """

VarSum = dict[CutVar, float]
"""
    A sum of variables is a dictionary where variables are associated to their linear coefficients.
"""

Containers = dict[float, Optional[int]]
""" Every container length associated to its available number (None for no upper bound). """

```

```python
Items = dict[float, int]
""" Every item length associated to the number of that item that needs to be packed. """

def calculate_residuals(containers: Containers, items: Items) -> set[float]:
    """ Calculates the set of residual lengths from the containers and items. """

    min_items = min(items)
    all_to_cut = list(containers)
    residuals = set()

    while len(all_to_cut) > 0:
        to_cut = all_to_cut[0]
        for item in items:
            if to_cut > item:
                cut = to_cut - item

                residuals.add(item)
                if cut >= min_items:
                    if cut not in residuals:
                        all_to_cut.append(cut)
                    residuals.add(cut)

        all_to_cut = all_to_cut[1:]

    return residuals

def merge_sums(sums: Iterable[VarSum]) -> VarSum:
    """ Sums many variable sums together. """

    final_sum: VarSum = {}
    for summation in sums:
        for variable, coefficient in summation.items():
```

```
60              final_coefficient = final_sum.get(variable)
61              final_sum[variable] = \
62                  coefficient if final_coefficient is None else final_coefficient + coefficient
63
64      # Remove variables with coefficient zero and with redundant names (e.g.: y7_5 and y7_2)
65      ret = {variable: coefficient for variable, coefficient in final_sum.items() if variable != 0}
66      return {(l, k): coefficient for (l, k), coefficient in ret.items() if \
67          not (k < l - k and (l, l - k) in ret)}
68
69  def output_variable(variable: CutVar, all_variables: set[str], items: Items) -> str:
70      """
71          Outputs LP code for a variable representing a cut. all_variables will be filled with all
72          variables ever outputted, so they can later be declared to be integers later on.
73      """
74
75      l, k = variable
76      ret = f'y{l}_{k}' if k in items else f'y{l}_{l - k}'
77      all_variables.add(ret)
78      return ret
79
80  def output_sum(summation: VarSum, all_variables: set[str], items: Items) -> str:
81      """
82          Outputs LP code with a sum of variables and their linear coefficients. all_variables will be
83          filled with allvariables ever outputted, so they can later be declared to be integers.
84      """
85
86      ret = ''
87      for variable, coefficient in sorted(summation.items(), key=lambda item: item[0], reverse=True):
88          str_coefficient = '' if coefficient == 1 else f'{coefficient} '
89          ret += f'{str_coefficient}{output_variable(variable, all_variables, items)} + '
90      return ret[:-3]
91
```

```
92  def output_objective(containers: Containers, \
93                       items: Items, \
94                       residuals: set[float], \
95                       all_variables: set[str]) -> str:
96      """
97          Outputs LP code with the objective function of the problem. all_variables will be filled
98          with all variables ever outputted, so they can later be declared to be integers.
99      """
100
101     ret = '/* Standardize optimizations of max{0, ...} */'
102     stock_residuals_union = set(containers.keys()).union(residuals)
103
104     for l in containers:
105         b_sum = {(k + l, k): -1 for k in items if k + l in stock_residuals_union}
106         c_sum = {(l, k): 1 for k in items if k < l}
107
108         ret += f'\nm{l} >= {output_sum(merge_sums([b_sum, c_sum]), all_variables, items)};'
109
110     objective = " + ".join([f'{l} m{l}' for l in containers])
111     return f'min: {objective};\n\n{ret}'
112
113 def output_balance_restrictions(containers: Containers, \
114                                 items: Items, \
115                                 residuals: set[float], \
116                                 all_variables: set[str]) -> str:
117     """
118         Outputs LP code with all balance restrictions for the problem. all_variables will be filled
119         with all variables ever outputted, so they can later be declared to be integers.
120     """
121
122     ret = '/* Balance restrictions */'
123     stock_residuals_union = set(containers.keys()).union(residuals)
```

```
    l_values = sorted(set(items.keys()).union(residuals).difference(set(containers.keys())))
    for l in l_values:
        a_set = {k for k in stock_residuals_union if k > l} if l in items else set()

        a_sum = {(k, l): 1 for k in a_set}
        b_sum = {(k + l, k): 1 for k in items if k + l in stock_residuals_union}
        c_sum = {(l, k): -1 for k in items if k < l}

        summation = merge_sums([a_sum, b_sum, c_sum])
        ret += f'\nl{l}: {output_sum(summation, all_variables, items)} >= {items.get(l, 0)};'

    return ret

def output_container_count_bounds(containers: Containers, \
                                  items: Items, \
                                  residuals: set[float], \
                                  all_variables: set[str]) -> str:
    """ Outputs LP code limiting the number of containers of each type to their upper bound. """

    ret = '/* Container upper bounds */'
    stock_residuals_union = set(containers.keys()).union(residuals)
    for l, upper_bound in containers.items():
        if upper_bound is not None:
            b_sum = {(k + l, k): -1 for k in items if k + l in stock_residuals_union}
            c_sum = {(l, k): 1 for k in items if k < l}
            summation = merge_sums([b_sum, c_sum])

            ret += f'\nc{l}: {output_sum(summation, all_variables, items)} <= {upper_bound};'

    return ret

def output_integer_restrictions(all_variables: set[str]) -> str:
```

```
    """ Outputs LP code telling that all variables ever outputted are integers. """
    return 'int ' + ', '.join(sorted(all_variables)) + ';'

def output_model(containers: Containers, items: Items) -> str:
    """ Outputs LP code modelling a give bin-packing problem. """

    ret = ''
    all_variables: set[str] = set()
    residuals = calculate_residuals(containers, items)

    ret += f'{output_objective(containers, items, residuals, all_variables)}\n\n'
    ret += f'{output_balance_restrictions(containers, items, residuals, all_variables)}\n\n'
    ret += f'{output_container_count_bounds(containers, items, residuals, all_variables)}\n\n'
    ret += f'{output_integer_restrictions(all_variables)}\n'
    return ret

if __name__ == '__main__':
    # Our problem's data
    print(output_model({11: None, 10: 5, 7: 5}, {2: 13, 4: 9, 5: 5}), end='')

    # Example from Dyckhoff's one-cut model:
    # print(output_model({5: None, 6: None, 9: None}, {2: 20, 3: 10, 4: 20}), end='')
```