Departamento de Engenharia Informática e de Sistemas

Metodologias de Otimização e Apoio à Decisão

Resolução de problemas de PL em Python

- Parte II -

EXEMPLO 2

Considere o seguinte problema:

"*No serviço de urgências de uma dada clínica, que funciona **24** horas por dia, os requerimentos mínimos de pessoal de enfermagem nas diferentes horas do dia são os seguintes:*

Horas	Requerimentos mínimos
0 - 4	**4**
4 - 8	**6**
8 - 12	**10**
12 - 16	**8**
16 - 20	**12**
20 - 0	**6**

*Cada enfermeiro trabalha **8** horas consecutivas por dia e os turnos iniciam-se de **4** em **4** horas a partir das **0** horas da madrugada.*

A clínica quer saber qual o número mínimo de enfermeiros que deve possuir nos seus quadros, para satisfazer os requisitos anteriores (considerando que cada enfermeiro trabalha num único turno e que esse turno é fixo)."

Desta forma, haverá **6** turnos diferentes, de **8** horas cada. Considerando $\mathbf{x_j}$ como o nº de enfermeiros a contratar para o turno **j**, o modelo matemático pode ser definido como se apresenta em seguida.

Minimizar $z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$ (Total de enfermeiros a contratar)

sujeito a

$x_6 + x_1 \geq 4$ (Mínimo de enfermeiros no período 0h-4h)

$x_1 + x_2 \geq 6$ (Mínimo de enfermeiros no período 4h-8h)

$x_2 + x_3 \geq 10$ (Mínimo de enfermeiros no período 8h-12h)

$x_3 + x_4 \geq 8$ (Mínimo de enfermeiros no período 12h-16h)

$x_4 + x_5 \geq 12$ (Mínimo de enfermeiros no período 16-20h)

$x_5 + x_6 \geq 6$ (Mínimo de enfermeiros no período 20h-0h)

$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0, x_6 \geq 0$

Para resolver este problema, escreve-se o código que se segue e que é bastante semelhante ao do EXEMPLO 1. A maior diferença é que se trata de um problema de minimização, o que precisa de ser indicado na criação do objeto *model* através da constante *LpMinimize*.

```
"""
Planeamento de turnos / Shift planning
"""

from pulp import *

# Criar modelo / Create model
model = LpProblem("Turnos-enfermeiros/Nurse-Shifts",LpMinimize)

# Definir variáveis de decisão / Define decision variables
x1=LpVariable("x1",0)
x2=LpVariable("x2",0)
x3=LpVariable("x3",0)
x4=LpVariable("x4",0)
x5=LpVariable("x5",0)
x6=LpVariable("x6",0)

# Adicionar função objetivo ao modelo / Add objective function to model
model += x1 + x2 + x3 + x4 + x5 + x6, "Total Enfermeiros/Nurses"

# Adicionar restrições ao modelo / Add constraints to the model
model += x6 + x1 >=  4, "Periodo/Period 00H00-04H00"
model += x1 + x2 >=  6, "Periodo/Period 04H00-08H00"
model += x2 + x3 >= 10, "Periodo/Period 08H00-12H00"
model += x3 + x4 >=  8, "Periodo/Period 12H00-16H00"
model += x4 + x5 >= 12, "Periodo/Period 16H00-20H00"
model += x5 + x6 >=  6, "Periodo/Period 20H00-00H00"

# Visualizar modelo / Visualize model
print("---------------------- Modelo / Model ----------------------")
print(model)

# Resolver modelo / Solve the model
model.solve()

# Mostrar resultados / Show results
print("------------------ Resultados / Results ------------------")
print(f"Estado / Status: {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {model.objective.value()}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name, constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```

Na sequência do código anterior, obtém-se os seguintes resultados:

```
In [1]: runfile('C:/Users/Teresa/Arquivos/MOAD/AULAS PRÁTICAS/
AULAS PRÁTICAS/PYTHON/Projetos')
--------------------- Modelo / Model ---------------------
Turnos-enfermeiros/Nurse-Shifts:
MINIMIZE
1*x1 + 1*x2 + 1*x3 + 1*x4 + 1*x5 + 1*x6 + 0
SUBJECT TO
Periodo/Period_00H00_04H00: x1 + x6 >= 4

Periodo/Period_04H00_08H00: x1 + x2 >= 6

Periodo/Period_08H00_12H00: x2 + x3 >= 10

Periodo/Period_12H00_16H00: x3 + x4 >= 8

Periodo/Period_16H00_20H00: x4 + x5 >= 12

Periodo/Period_20H00_00H00: x5 + x6 >= 6

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
x4 Continuous
x5 Continuous
x6 Continuous

------------------ Resultados / Results ------------------
Estado / Status: 1 <=> Optimal
z* = 26.0
x1* = 4.0
x2* = 2.0
x3* = 8.0
x4* = 6.0
x5* = 6.0
x6* = 0.0
Periodo/Period_00H00_04H00: 0.0
Periodo/Period_04H00_08H00: 0.0
Periodo/Period_08H00_12H00: 0.0
Periodo/Period_12H00_16H00: 6.0
Periodo/Period_16H00_20H00: 0.0
Periodo/Period_20H00_00H00: 0.0
```

Interpretação dos resultados:

A clínica deverá contratar um total de **26** enfermeiros que serão necessários para assegurar os requerimentos nos vários períodos horários. Destes, **4** trabalharão no turno 1, **2** no turno 2, **8** no turno 3, **6** no turno 4 e **6** no turno 5. O turno 6 não precisará de existir. Verifica-se que em todos os períodos horários, o nº de enfermeiros a trabalhar será igual ao mínimo exigido, à exceção do período das 12:00 às 16:00, em que estarão presentes mais 6 enfermeiros do que era requerido.

EXEMPLO 3

Considere-se o seguinte exemplo de gestão de recursos (adaptado de *https://realpython.com/linear-programming-python/#resource-allocation-problem*).

"*Determinada fábrica produz quatro tipos de produtos diferentes (**P_1**, **P_2**, **P_3** e **P_4**), usando as matérias-primas **A** e **B**. Sendo **x_1**, **x_2**, **x_3** e **x_4**, as quantidades a produzir diariamente de cada um dos produtos, o objetivo é determinar o esquema de produção diária que maximiza o lucro total. Sabe-se que o lucro por cada unidade de produto é de **€ 20**, **€ 12**, **€ 40** e **€ 25**, para **P_1**, **P_2**, **P_3** e **P_4**, respetivamente. Devido a restrições de mão-de-obra, o número total de unidades produzidas por dia não pode exceder **50**. No processo de produção: por cada unidade de **P_1**, são consumidas três unidades da matéria-prima **A**; cada unidade de **P_2** requer duas unidades da matéria-prima **A** e uma unidade da matéria-prima **B**; cada unidade de **P_3** precisa de uma unidade de **A** e de duas unidades de **B**; finalmente, cada unidade de **P_4** requer três unidades de **B**. Devido às restrições de transporte e armazenamento, a fábrica pode dispor de um máximo de cem unidades da matéria-prima **A** e de noventa unidades de matéria-prima **B**, por dia.*"

Deste modo, o modelo matemático pode ser definido como se apresenta em seguida.

Maximizar $z = 20\,x_1 + 12\,x_2 + 40\,x_3 + 25\,x_4$ (Lucro diário)

sujeito a

$$x_1 + x_2 + x_3 + x_4 \leq 50 \quad \text{(Mão-de-obra)}$$
$$3\,x_1 + 2\,x_2 + x_3 \leq 100 \quad \text{(Matéria-prima A)}$$
$$x_2 + 2\,x_3 + 3\,x_4 \leq 90 \quad \text{(Matéria-prima B)}$$
$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Para resolver este problema, o código é bastante semelhante ao dos exemplos anteriores, no entanto, recorre-se a um dicionário (**x**) para armazenar todas as variáveis de decisão.

Embora não sendo este o caso, quando o número de variáveis é elevado, esta abordagem torna-se vantajosa, uma vez que os dicionários permitem armazenar os nomes ou índices das variáveis de decisão como chaves, e os correspondentes objetos *LpVariable*, como valores.

```
"""
Gestão de Recursos / Resource Management
"""
from pulp import *

# Criar modelo / Create model
model = LpProblem("Gestao-de-Recursos/Resource-Management",LpMaximize)

# Definir variáveis de decisão / Define decision variables
x = {j: LpVariable(name=f"x{j}", lowBound=0) for j in range(1, 5)}

# Adicionar função objetivo ao modelo / Add objective function to model
model += 20 * x[1] + 12 * x[2] + 40 * x[3] + 25 * x[4]

# Adicionar restrições ao modelo / Add constraints to model
model += x[1] + x[2] + x[3] + x[4] <= 50, "mao-de-obra/hand-labor"

model += 3 * x[1] + 2 * x[2] + x[3] <= 100, "materia-prima-A/raw-material-A"

model += x[2] + 2 * x[3] + 3 * x[4] <= 90, "materia-prima-B/raw-material-B"

# Visualizar modelo / Visualize model
print("---------------------- Modelo / Model ----------------------")
print(model)

# Resolver modelo / Solve model
model.solve()

# Mostrar resultados / Show results
print("------------------- Resultados / Results -------------------")
print(f"Estado / Status: {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {model.objective.value()}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name, constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```

Ao contrário dos exemplos anteriores em que as variáveis de decisão eram criadas individualmente e o acesso às mesmas era direto, no código anterior estas são acedidas através da indexação do dicionário com a chave que é o índice dessas mesmas variáveis (1, 2, 3 e 4).

Na sequência do bloco de instruções apresentado, obtém-se os seguintes resultados:

```
In [4]: runfile('C:/Users/Teresa/Arquivos/MOAD/AULAS PRÁTICAS/
AULAS PRÁTICAS/PYTHON/Projetos')
---------------------- Modelo / Model ----------------------
Gestao-de-Recursos/Resource-Management:
MAXIMIZE
20*x1 + 12*x2 + 40*x3 + 25*x4 + 0
SUBJECT TO
mao_de_obra/hand_labor: x1 + x2 + x3 + x4 <= 50

materia_prima_A/raw_material_A: 3 x1 + 2 x2 + x3 <= 100

materia_prima_B/raw_material_B: x2 + 2 x3 + 3 x4 <= 90

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
x4 Continuous

------------------ Resultados / Results ------------------
Estado / Status: 1 <=> Optimal
z* = 1900.0
x1* = 5.0
x2* = 0.0
x3* = 45.0
x4* = 0.0
mao_de_obra/hand_labor: 0.0
materia_prima_A/raw_material_A: -40.0
materia_prima_B/raw_material_B: 0.0
```

Interpretação dos resultados:

De acordo com os valores obtidos para as variáveis de decisão e função objetivo, a empresa deve produzir diariamente **5** unidades do produto P1 e **45** unidades do produto P3, conseguindo, dessa forma, um lucro máximo de **1900** € por dia. Dados os valores das restrições, podemos concluir que a mão-de-obra será usada na sua totalidade, bem como a matéria-prima B. Da matéria-prima A, sobrarão 40 unidades.

Em relação ao código anterior, existe a alternativa de definir os parâmetros do modelo como vetores/matrizes (listas e listas de listas) e tornar a construção da função objetivo e restrições, mais genérica.

Outra alternativa para tornar o código ainda mais genérico, pode-se pedir esses parâmetros ao utilizador.

Os códigos que se apresentam em seguida ilustram essas duas possibilidades.

1ª alternativa:

```
"""
Gestão de Recursos / Resource Management
"""
from pulp import *

# Definir parâmetros / Define parameters cj, aij, bi
c=[20, 12, 40, 25]
a=[[1,1,1,1],[3,2,1,0],[0,1,2,3]]
b=[50,100,90]

# Criar modelo / Create model
model = LpProblem("Gestao-de-Recursos/Resource-Management",LpMaximize)

# Definir variáveis de decisão / Define decision variables
x = {j: LpVariable(name=f"x{j}", lowBound=0) for j in range(1,5)}

# Adicionar função objetivo ao modelo / Add objective function to model
model += lpSum([c[j] * x[j+1]] for j in range(4))

# Adicionar restrições ao modelo / Add constraints to model
for i in range(3):
      model += lpSum([a[i][j] * x[j+1]] for j in range(4)) <= b[i]

# Visualizar modelo / Visualize model
print("---------------------- Modelo / Model ----------------------")
print(model)

# Resolver modelo / Solve model
model.solve()

# Mostrar resultados / Show results
print("------------------ Resultados / Results ------------------")
print(f"Estado / Status: {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {model.objective.value()}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name, constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```

2ª alternativa:

```
"""
Gestão de Recursos / Resource Management
"""
from pulp import *

# Pedir parâmetros / Ask for parameters cj, aij, bi
n=int(input("Insert number of variables:"))
m = int(input("Insert number of contraints: "))

c=[]
for j in range(1,n+1):
    element=float((input(f"Insert coefficient c{j}:")))
    c.append(element)

a=[]
for i in range(1,m+1):
    row = [float(input(f"Insert coefficient a({i},{j}):")) for j in range(1,n+1)]
    a.append(row)

b=[]
for i in range(1,m+1):
    element=float((input(f"Insert coefficient b{i}:")))
    b.append(element)


# Criar modelo / Create model
model = LpProblem("Gestao-de-Recursos/Resource-Management",LpMaximize)

# Definir variáveis de decisão / Define decision variables
x = {j: LpVariable(name=f"x{j}", lowBound=0) for j in range(1, n+1)}

# Adicionar função objetivo ao modelo / Add objective function to model
model += lpSum([c[j] * x[j+1]] for j in range(n))


# Adicionar restrições ao modelo / Add constraints to model
for i in range(3):
      model += lpSum([a[i][j] * x[j+1]] for j in range(n)) <= b[i]

# Visualizar modelo / Visualize model
print("---------------------- Modelo / Model ----------------------")
print(model)

# Resolver modelo / Solve model
model.solve()

# Mostrar resultados / Show results
print("------------------ Resultados / Results ------------------")
print(f"Estado / Status: {model.status} <=> {LpStatus[model.status]}")
print(f"z* = {model.objective.value()}")
for var in model.variables():
    print(f"{var.name}* = {var.value()}")
for name, constraint in model.constraints.items():
    print(f"{name}: {constraint.value()}")
```