# problema_DESIG

February 15, 2025

## 1 Problema da Designação

[18]:

```
import cplex
import networkx as nx
import matplotlib.pyplot as plt
import string
```

### 1.1 Leitura e e pré-processamento dos dados

[19]:

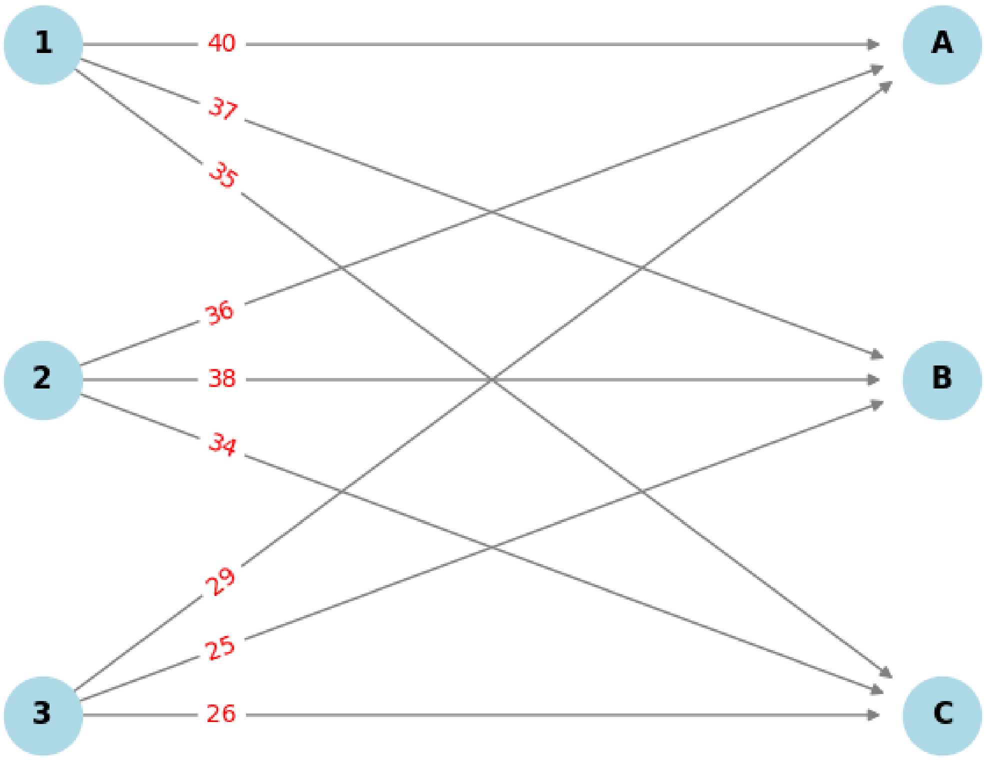
```
file = "in_desig.txt"

with open(file, 'r') as f:
    lines = f.readlines()
    lines = [line.strip() for line in lines]
    lines = list(filter(None, lines))

num_nodes, num_edges = map(int, lines[0].strip().split())

pessoas = []
tarefas = []
letras = string.ascii_uppercase  # 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for linha in lines[1:num_nodes+1]:
    node_id, tipo = map(int, linha.split())
    if tipo == 0:
        pessoas.append(node_id)
    elif tipo == 1:
        tarefas.append(node_id)

costs = [[0 for _ in range(len(tarefas))] for _ in range(len(pessoas))]
for line in lines[num_nodes+1:]:
    partes = line.split()
    if len(partes) >= 3:
        source, target, cost = int(partes[0]), int(partes[1]), int(partes[2])
        # Verifica se a origem é uma pessoa e o destino é uma tarefa
        if source in pessoas and target in tarefas:
            # Obtém o índice da pessoa e da tarefa na ordem definida
            i = pessoas.index(source)
            j = tarefas.index(target)
            costs[i][j] = cost

print("Pessoas:", pessoas)
print("Tarefas:", tarefas)
print("Matriz de Custos:")
for line in costs:
    print(" ".join(map(str, line)))
```

```
Pessoas: [0, 1, 2]
Tarefas: [3, 4, 5]
Matriz de Custos:
40 37 35
36 38 34
29 25 26
```

## 1.2 Visualização do Problema

[20]:

```
G = nx.DiGraph()

labels = {}
for idx, p in enumerate(pessoas):
    node_label = pessoas[idx] + 1
    G.add_node(p, bipartite=0, label=node_label)
    labels[p] = node_label

for idx, t in enumerate(tarefas):
    node_label = letras[idx]
    G.add_node(t, bipartite=1, label=node_label)
    labels[t] = node_label

for i, p in enumerate(pessoas):
    for j, t in enumerate(tarefas):
        if costs[i][j] != 0:
            G.add_edge(p, t, weight=costs[i][j])

pos = {}
for idx, p in enumerate(pessoas):
    pos[p] = (0, -idx)
for idx, t in enumerate(tarefas):
    pos[t] = (1, -idx)

edge_labels = nx.get_edge_attributes(G, 'weight')

plt.figure(figsize=(8, 6))
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=1000)
nx.draw_networkx_labels(G, pos, labels=labels, font_size=12, font_weight='bold')
nx.draw_networkx_edges(
    G, pos,
    arrowstyle='-|>',
    arrowsize=10,
    min_target_margin = 25,
    edge_color='gray'
    )
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red',␣
 ↪label_pos=0.8)

plt.axis('off')
plt.show()
```



### 1.3 Modelagem e Solução

[21]:

```
model = cplex.Cplex()
model.objective.set_sense(model.objective.sense.minimize)

n = len(pessoas)
```

```python
variaveis = [f"x{i+1}{letras[j]}" for i in range(n) for j in range(n)]
num_variaveis = len(variaveis)
lb = [0.0] * num_variaveis
ub = [1.0] * num_variaveis
types = [model.variables.type.binary] * num_variaveis

# Coeficientes da função objetivo: custo de cada atribuição
objetivo = [costs[i][j] for i in range(n) for j in range(n)]

# Definir as variáveis: binárias (0 ou 1)
model.variables.add(names=variaveis, obj=objetivo, lb=lb, ub=ub, types=types)

# Restrição 1: Cada trabalhador é designado a exatamente um trabalho
for i in range(n):
    indices = [f"x{i+1}{letras[j]}" for j in range(n)]
    coef = [1.0] * n
    model.linear_constraints.add(
        lin_expr=[cplex.SparsePair(ind=indices, val=coef)],
        senses=["E"],
        rhs=[1.0],
        names=[f"pessoa_{i}"]
    )

# Restrição 2: Cada trabalho é designado a exatamente um trabalhador
for j in range(n):
    indices = [f"x{i+1}{letras[j]}" for i in range(n)]
    coef = [1.0] * n
    model.linear_constraints.add(
        lin_expr=[cplex.SparsePair(ind=indices, val=coef)],
        senses=["E"],
        rhs=[1.0],
        names=[f"tarefa_{letras[j]}"]
    )

%time model.solve()
```

```
Version identifier: 22.1.0.0 | 2022-03-25 | 54982fbec
CPXPARAM_Read_DataCheck                          1
Found incumbent of value 104.000000 after 0.00 sec. (0.00 ticks)
Found incumbent of value 99.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
CPXPARAM_Read_DataCheck                          1
Found incumbent of value 104.000000 after 0.00 sec. (0.00 ticks)
Found incumbent of value 99.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 6 rows, 9 columns, and 18 nonzeros.
Reduced MIP has 9 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Presolve time = 0.01 sec. (0.01 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 6 rows, 9 columns, and 18 nonzeros.
Reduced MIP has 9 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.01 sec. (0.01 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Clique table members: 6.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time = 0.00 sec. (0.01 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                           99.0000        0.0000           100.00%
*     0     0      integral     0       96.0000       96.0000        1    0.00%
Elapsed time = 0.05 sec. (0.05 ticks, tree = 0.00 MB, solutions = 2)

Root node processing (before b&c):
  Real time             =    0.06 sec. (0.05 ticks)
Parallel b&c, 4 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.06 sec. (0.05 ticks)
CPU times: user 79.3 ms, sys: 11.6 ms, total: 90.9 ms
Wall time: 71.1 ms
```

## 1.4 Sumário dos resultados

[22]:

```python
print("Status da solução:", model.solution.get_status_string())
print(f"Custo total: {model.solution.get_objective_value()}")
assignment = {}
sol_values = model.solution.get_values()
for i in range(n):
    for j in range(n):
        idx = i * n + j
        if sol_values[idx] > 0.5:  # Considera 1 se > 0.5
            assignment[i] = j
            print(f"Pessoa {i+1} designada a tarefa {letras[j]}")

model.write("./output/model_desig.lp")
model.solution.write("./output/solution_desig.sol")
```

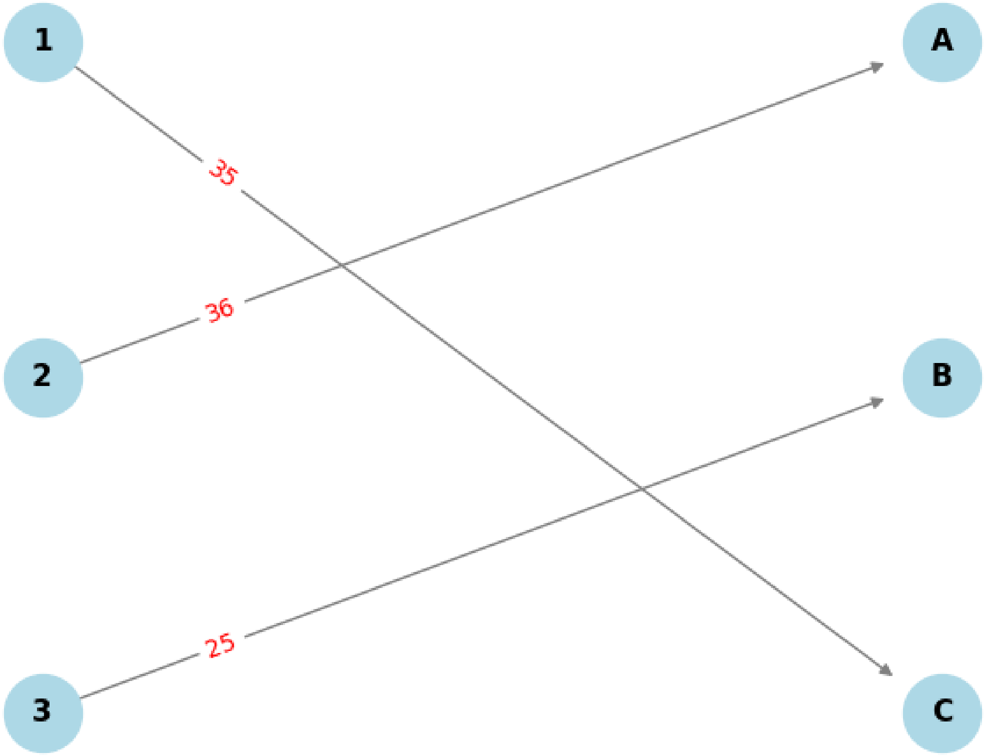
```
assignment
```

```
Status da solução: integer optimal solution
Custo total: 96.0
Pessoa 1 designada a tarefa C
Pessoa 2 designada a tarefa A
Pessoa 3 designada a tarefa B
```

[22]:
```
{0: 2, 1: 0, 2: 1}
```

[23]:
```
G = nx.DiGraph()

labels = {}
for idx, p in enumerate(pessoas):
    node_label = pessoas[idx] + 1
    G.add_node(p, bipartite=0, label=node_label)
    labels[p] = node_label

for idx, t in enumerate(tarefas):
    node_label = letras[idx]
    G.add_node(t, bipartite=1, label=node_label)
    labels[t] = node_label

for i, p in enumerate(assignment):
    G.add_edge(pessoas[i], tarefas[assignment[p]],␣
  ↪weight=costs[i][assignment[p]])

pos = {}
for idx, p in enumerate(pessoas):
    pos[p] = (0, -idx)
for idx, t in enumerate(tarefas):
    pos[t] = (1, -idx)

edge_labels = nx.get_edge_attributes(G, 'weight')

plt.figure(figsize=(8, 6))
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=1000)
nx.draw_networkx_labels(G, pos, labels=labels, font_size=12, font_weight='bold')
nx.draw_networkx_edges(
    G, pos,
    arrowstyle='-|>',
    arrowsize=10,
    min_target_margin = 25,
    edge_color='gray'
    )
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red',␣
  ↪label_pos=0.8)
```

```
plt.axis('off')
plt.show()
```


1
A
35
36
2
B
25
3
C