

# experimento 7: 50 x 200 rodou - 1294838d18

## USER

A base de dados anexada possui quatro variáveis: IDENTIFICADOR, CARGO, PERCENT\_PRI E DISTANCE\_KM. A primeira variável é o código que identifica o funcionário, a segunda apresenta os códigos dos cargos, a terceira é um índice que varia de 0 a 1, sendo o 0 o melhor e, por fim, a última variável é a distância em KM entre o funcionário e o cargo que ele poderá assumir, ou seja, quanto maior a distância, pior. Gostaria que me propusesse uma metodologia para solucionar meu problema de alocação de forma a minimizar o PERCENT\_PRI ou a DISTANCE\_KM ou os dois juntos.

## BOT

Para resolver o problema de alocação de funcionários aos cargos minimizando **PERCENT\_PRI** (índice de prioridade) e/ou **DISTANCE\_KM** (distância em quilômetros), podemos adotar uma abordagem de **otimização combinatória**. Abaixo, apresento uma metodologia passo a passo para abordar o problema:

---

### 1. Definição do Problema

O problema pode ser formulado como uma **alocação ótima** onde:

- Cada funcionário (IDENTIFICADOR) deve ser alocado a um cargo (CARGO).
- O objetivo é minimizar:
  - **PERCENT\_PRI**: Quanto menor o índice, melhor a alocação.
  - **DISTANCE\_KM**: Quanto menor a distância, melhor a alocação.
  - Ou uma combinação ponderada de ambos.

Este problema pode ser modelado como um **problema de otimização de atribuição**.

---

### 2. Modelagem Matemática

Podemos modelar o problema como um problema de programação linear inteira (ILP). A formulação é a seguinte:

#### Variáveis de Decisão

Definimos uma variável binária:

- $x_{ij} = 1$  se o funcionário  $i$  for alocado ao cargo  $j$ , e  $0$  caso contrário.

#### Função Objetivo

A função objetivo depende do critério de otimização:

1. **Minimizar apenas PERCENT\_PRI**:

$$\text{Minimizar: } \sum_{i,j} \text{PERCENT\_PRI}_{ij} \cdot x_{ij}$$

2. **Minimizar apenas DISTANCE\_KM**:

$$\text{Minimizar: } \sum_{i,j} \text{DISTANCE\_KM}_{ij} \cdot x_{ij}$$

3. **Minimizar ambos (com pesos  $w_1$  e  $w_2$ )**:

$$\text{Minimizar: } \sum_{i,j} \left( w_1 \cdot \text{PERCENT\_PRI}_{ij} + w_2 \cdot \text{DISTANCE\_KM}_{ij} \right) \cdot x_{ij}$$

Onde  $w_1$  e  $w_2$  são pesos que refletem a importância relativa de cada critério.

#### Restrições

1. **Cada funcionário é alocado a exatamente um cargo**:

$$\sum_j x_{ij} = 1 \quad \forall i$$

2. **Cada cargo pode ser ocupado por no máximo um funcionário**:

$$\sum_i x_{ij} \leq 1 \quad \forall j$$

### 3. **\*\*Variáveis binárias\*\***:

```
\[
x_{ij} \in \{0, 1\} \quad \forall i, j
\]
```

---

### ### **\*\*3. Escolha do Método de Solução\*\***

O problema pode ser resolvido utilizando métodos de otimização combinatória. Algumas opções incluem:

#### #### **\*\*a. Algoritmos Exatos\*\***

- **\*\*Método de Programação Linear Inteira (ILP)\*\***:

- Utiliza solvers como **\*\*CPLEX\*\***, **\*\*Gurobi\*\***, ou **\*\*SCIP\*\*** para encontrar a solução ótima.
- Adequado para problemas de tamanho pequeno a médio.

#### #### **\*\*b. Algoritmos Aproximados\*\***

- **\*\*Algoritmos Heurísticos\*\***:

- **\*\*Algoritmo Húngaro\*\***: Ideal para problemas de atribuição simples (minimização de um único critério).
- **\*\*Simulated Annealing\*\*** ou **\*\*Algoritmos Genéticos\*\***: Úteis para problemas maiores ou com múltiplos critérios.

- **\*\*Metaheurísticas Multiobjetivo\*\***:

- **\*\*NSGA-II\*\*** (Non-dominated Sorting Genetic Algorithm): Para otimizar múltiplos objetivos simultaneamente.

#### #### **\*\*c. Algoritmos de Programação Multiobjetivo\*\***

- Se for necessário otimizar **\*\*PERCENT\_PRI\*\*** e **\*\*DISTANCE\_KM\*\*** simultaneamente, pode-se usar:

- **\*\*Método de Ponderação\*\***: Combina os dois objetivos em uma única função com pesos  $(w_1)$  e  $(w_2)$ .
- **\*\*Método de Fronteira de Pareto\*\***: Encontra um conjunto de soluções ótimas (não-dominadas) que equilibram os dois critérios.

---

### ### **\*\*4. Implementação Prática\*\***

#### #### **\*\*Passo 1: Preparação dos Dados\*\***

- Organize os dados em uma matriz onde cada linha representa um funcionário e cada coluna representa um cargo.
- As entradas da matriz são os valores de **\*\*PERCENT\_PRI\*\*** ou **\*\*DISTANCE\_KM\*\*** (ou uma combinação ponderada).

#### #### **\*\*Passo 2: Escolha do Solver\*\***

- Para problemas pequenos:

- Use bibliotecas como **\*\*PuLP\*\*** ou **\*\*OR-Tools\*\*** (Python) para resolver o problema de ILP.

- Para problemas grandes:

- Considere heurísticas ou metaheurísticas.

#### #### **\*\*Passo 3: Implementação\*\***

Aqui está um exemplo de implementação em Python usando **\*\*PuLP\*\*** para minimizar ambos os critérios:

```
```python
from pulp import LpProblem, LpMinimize, LpVariable, lpSum
```

# Dados de exemplo

funcionarios = [1, 2, 3] # IDs dos funcionários

cargos = [1, 2, 3] # IDs dos cargos

percent\_pri = { # PERCENT\_PRI[i][j]

(1, 1): 0.2, (1, 2): 0.4, (1, 3): 0.6,

(2, 1): 0.3, (2, 2): 0.1, (2, 3): 0.5,

(3, 1): 0.7, (3, 2): 0.8, (3, 3): 0.2

}

distance\_km = { # DISTANCE\_KM[i][j]

(1, 1): 10, (1, 2): 20, (1, 3): 30,

(2, 1): 15, (2, 2): 25, (2, 3): 35,

(3, 1): 5, (3, 2): 10, (3, 3): 15

}

# Pesos para os critérios

w1 = 0.5 # Peso para PERCENT\_PRI

w2 = 0.5 # Peso para DISTANCE\_KM

# Modelo de otimização

model = LpProblem("AlocacaoFuncionarios", LpMinimize)

```

# Variáveis de decisão
x = LpVariable.dicts("x", [(i, j) for i in funcionarios for j in cargos], cat="Binary")

# Função objetivo: Minimizar PERCENT_PRI e DISTANCE_KM
model += lpSum((w1 * percent_pri[i, j] + w2 * distance_km[i, j]) * x[i, j] for i in funcionarios for j in cargos)

# Restrições: Cada funcionário é alocado a exatamente um cargo
for i in funcionarios:
    model += lpSum(x[i, j] for j in cargos) == 1

# Restrições: Cada cargo é ocupado por no máximo um funcionário
for j in cargos:
    model += lpSum(x[i, j] for i in funcionarios) <= 1

# Resolver o modelo
model.solve()

# Resultados
for i in funcionarios:
    for j in cargos:
        if x[i, j].value() == 1:
            print(f"Funcionário {i} alocado ao cargo {j}")
    ...

---

### **5. Validação e Ajustes**
- **Validação**: Teste a solução com diferentes pesos ( $w_1$  e  $w_2$ ) para avaliar o impacto na alocação.
- **Ajustes**: Se necessário, inclua restrições adicionais, como:
    - Limites máximos de distância.
    - Priorização de cargos específicos para determinados funcionários.

---

### **6. Conclusão**
Essa metodologia permite alocar funcionários aos cargos de forma eficiente, minimizando PERCENT_PRI, DISTANCE_KM, ou ambos. A escolha do método de solução depende do tamanho do problema e da necessidade de otimização multiobjetivo.

```

Execution Result:

```

...

Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /opt/venv/lib/python3.12/site-packages/pulp/solverdir/cbc/linux/64/cbc
/tmp/3d775cf6da6e40a49c077e1eb25899fc-pulp.mps -timeMode elapsed -branch -printingOptions all -solution
/tmp/3d775cf6da6e40a49c077e1eb25899fc-pulp.sol (default strategy 1)
At line 2 NAME      MODEL
At line 3 ROWS
At line 11 COLUMNS
At line 57 RHS
At line 64 BOUNDS
At line 74 ENDATA
Problem MODEL has 6 rows, 9 columns and 18 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 25.25 - 0.00 seconds
Cgl0004I processed model has 6 rows, 9 columns (9 integer (9 of which binary)) and 18 elements
Cutoff increment increased from 1e-05 to 0.04995
Cbc0038I Initial state - 0 integers unsatisfied sum - 0
Cbc0038I Solution found of 25.25
Cbc0038I Before mini branch and bound, 9 integers at bound fixed and 0 continuous
Cbc0038I Mini branch and bound did not improve solution (0.00 seconds)
Cbc0038I After 0.00 seconds - Feasibility pump exiting with objective of 25.25 - took 0.00 seconds
Cbc0012I Integer solution of 25.25 found by feasibility pump after 0 iterations and 0 nodes (0.00 seconds)
Cbc0001I Search completed - best objective 25.25, took 0 iterations and 0 nodes (0.00 seconds)

```

Cbc00351 Maximum depth 0, 0 variables fixed on reduced cost

Cuts at root node changed objective from 25.25 to 25.25

Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value: 25.25000000

Enumerated nodes: 0

Total iterations: 0

Time (CPU seconds): 0.00

Time (Wallclock seconds): 0.00

Option for printingOptions changed from normal to all

Total time (CPU seconds): 0.00 (Wallclock seconds): 0.00

Funcionário 1 alocado ao cargo 1

Funcionário 2 alocado ao cargo 2

Funcionário 3 alocado ao cargo 3

``