

# experimento 6: claude - 14a70b55e8

## USER

A base de dados anexada no arquivo dados\_50\_50.csv 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. Determine e execute o código para solucionar o problema e indique o custo total minimizado.

## BOT

Vou ajudar você a resolver este problema de otimização passo a passo.

1) Primeiro, vamos entender o problema:

- Temos 50 funcionários (IDENTIFICADOR)
- Cada funcionário pode ser alocado em diferentes cargos
- Temos dois critérios: PERCENT\_PRI (quanto menor, melhor) e DISTANCE\_KM (quanto menor, melhor)
- Precisamos alocar cada funcionário em um cargo minimizando estes critérios

2) Abordagem proposta:

- Usar programação linear para otimização
- Como temos dois objetivos (PERCENT\_PRI e DISTANCE\_KM), podemos:
  - a) Normalizar ambos os critérios e criar uma função objetivo única
  - b) Dar pesos iguais para ambos os critérios

3) Vamos implementar usando Python com a biblioteca PuLP:

```
```python
import pandas as pd
import pulp
import numpy as np

# Ler os dados
df = pd.read_csv('dados_50_50.csv', sep=';', decimal=',')

# Normalizar as variáveis
df['PERCENT_PRI_NORM'] = (df['PERCENT_PRI'] - df['PERCENT_PRI'].min()) / (df['PERCENT_PRI'].max() - df['PERCENT_PRI'].min())
df['DISTANCE_KM_NORM'] = (df['DISTANCE_KM'] - df['DISTANCE_KM'].min()) / (df['DISTANCE_KM'].max() - df['DISTANCE_KM'].min())

# Criar função objetivo combinada (peso igual para ambos os critérios)
df['COST'] = 0.5 * df['PERCENT_PRI_NORM'] + 0.5 * df['DISTANCE_KM_NORM']

# Criar o problema de otimização
prob = pulp.LpProblem("Employee_Assignment", pulp.LpMinimize)

# Criar variáveis de decisão
choices = pulp.LpVariable.dicts("Choice",
                                ((i, j) for i in df['IDENTIFICADOR'].unique()
                                 for j in df['CARGO'].unique()),
                                cat='Binary')

# Função objetivo
prob += pulp.lpSum(choices[i, j] * df[(df['IDENTIFICADOR']==i) & (df['CARGO']==j)]['COST']).iloc[0]
                    for i in df['IDENTIFICADOR'].unique()
                    for j in df['CARGO'].unique())

# Restrições
# Cada funcionário deve ser alocado a exatamente um cargo
```

```

for i in df['IDENTIFICADOR'].unique():
    prob += pulp.lpSum(choices[i,j] for j in df['CARGO'].unique()) == 1

# Cada cargo deve receber exatamente um funcionário
for j in df['CARGO'].unique():
    prob += pulp.lpSum(choices[i,j] for i in df['IDENTIFICADOR'].unique()) == 1

# Resolver o problema
prob.solve()

# Calcular custo total
total_cost = pulp.value(prob.objective)

print(f"Status: {pulp.LpStatus[prob.status]}")
print(f"Custo total minimizado: {total_cost}")

# Mostrar alocações
print("\nAlocações:")
for i in df['IDENTIFICADOR'].unique():
    for j in df['CARGO'].unique():
        if pulp.value(choices[i,j]) == 1:
            row = df[(df['IDENTIFICADOR']==i) & (df['CARGO']==j)].iloc[0]
            print(f"Funcionário {i} -> Cargo {j} (PERCENT_PRI: {row['PERCENT_PRI']}, DISTANCE_KM: {row['DISTANCE_KM']})")
    ...

```

#### 4) Este código:

- Normaliza ambas as métricas para ficarem na mesma escala (0-1)
- Cria uma função objetivo combinada com peso igual para ambos os critérios
- Usa programação linear inteira binária para encontrar a alocação ótima
- Garante que cada funcionário é alocado a exatamente um cargo e vice-versa
- Minimiza o custo total considerando tanto PERCENT\_PRI quanto DISTANCE\_KM

#### 5) Benefícios desta abordagem:

- Considera ambos os critérios de forma balanceada
- Garante uma solução ótima global
- É flexível (podemos ajustar os pesos dos critérios se necessário)
- Garante que todas as restrições são atendidas

#### 6) Limitações:

- Pode ser computacionalmente intensivo para problemas muito grandes
- Assume que os critérios têm importância igual (pode ser ajustado se necessário)

O código acima fornecerá a alocação ótima de funcionários aos cargos, minimizando tanto o PERCENT\_PRI quanto a DISTANCE\_KM de forma balanceada. O custo total minimizado será apresentado, junto com as alocações específicas de cada funcionário.

#### 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/c1944ef3ea9d4c05aad65eb2c090d1a0-pulp.mps -timeMode elapsed -branch -printingOptions all -solution
/tmp/c1944ef3ea9d4c05aad65eb2c090d1a0-pulp.sol (default strategy 1)
At line 2 NAME      MODEL
At line 3 ROWS
At line 105 COLUMNS
At line 12606 RHS
At line 12707 BOUNDS
At line 15208 ENDATA
Problem MODEL has 100 rows, 2500 columns and 5000 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 28.8931 - 0.00 seconds
Cgl0004I processed model has 100 rows, 2500 columns (2500 integer (2500 of which binary)) and 5000 elements

```

Cbc00381 Initial state - 0 integers unsatisfied sum - 0  
Cbc00381 Solution found of 28.8931  
Cbc00381 Before mini branch and bound, 2500 integers at bound fixed and 0 continuous  
Cbc00381 Mini branch and bound did not improve solution (0.06 seconds)  
Cbc00381 After 0.06 seconds - Feasibility pump exiting with objective of 28.8931 - took 0.00 seconds  
Cbc00121 Integer solution of 28.893106 found by feasibility pump after 0 iterations and 0 nodes (0.06 seconds)  
Cbc00011 Search completed - best objective 28.89310555618498, took 0 iterations and 0 nodes (0.06 seconds)  
Cbc00351 Maximum depth 0, 0 variables fixed on reduced cost  
Cuts at root node changed objective from 28.8931 to 28.8931  
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: 28.89310556  
Enumerated nodes: 0  
Total iterations: 0  
Time (CPU seconds): 0.05  
Time (Wallclock seconds): 0.06

Option for printingOptions changed from normal to all  
Total time (CPU seconds): 0.05 (Wallclock seconds): 0.07

Status: Optimal  
Custo total minimizado: 28.893105556184697

Alocações:

Funcionário 1 -> Cargo 10 (PERCENT\_PRI: 0.85417, DISTANCE\_KM: 1414.13)  
Funcionário 2 -> Cargo 6 (PERCENT\_PRI: 0.5625, DISTANCE\_KM: 2934.83)  
Funcionário 3 -> Cargo 22 (PERCENT\_PRI: 0.65957, DISTANCE\_KM: 3415.07)  
Funcionário 4 -> Cargo 16 (PERCENT\_PRI: 0.42553, DISTANCE\_KM: 1758.24)  
Funcionário 5 -> Cargo 69 (PERCENT\_PRI: 0.6383, DISTANCE\_KM: 789.76)  
Funcionário 6 -> Cargo 3 (PERCENT\_PRI: 1.0, DISTANCE\_KM: 3355.79)  
Funcionário 7 -> Cargo 70 (PERCENT\_PRI: 0.83333, DISTANCE\_KM: 742.76)  
Funcionário 8 -> Cargo 11 (PERCENT\_PRI: 0.97872, DISTANCE\_KM: 1604.7)  
Funcionário 9 -> Cargo 57 (PERCENT\_PRI: 0.85106, DISTANCE\_KM: 670.8)  
Funcionário 10 -> Cargo 13 (PERCENT\_PRI: 1.0, DISTANCE\_KM: 3590.74)  
Funcionário 11 -> Cargo 71 (PERCENT\_PRI: 0.42553, DISTANCE\_KM: 1394.51)  
Funcionário 12 -> Cargo 30 (PERCENT\_PRI: 0.0625, DISTANCE\_KM: 93.51)  
Funcionário 13 -> Cargo 1 (PERCENT\_PRI: 0.2766, DISTANCE\_KM: 1575.76)  
Funcionário 14 -> Cargo 29 (PERCENT\_PRI: 0.97368, DISTANCE\_KM: 3266.16)  
Funcionário 15 -> Cargo 56 (PERCENT\_PRI: 0.68085, DISTANCE\_KM: 1487.11)  
Funcionário 16 -> Cargo 64 (PERCENT\_PRI: 0.97872, DISTANCE\_KM: 1830.77)  
Funcionário 17 -> Cargo 61 (PERCENT\_PRI: 0.97872, DISTANCE\_KM: 1394.51)  
Funcionário 18 -> Cargo 9 (PERCENT\_PRI: 0.89474, DISTANCE\_KM: 2991.58)  
Funcionário 19 -> Cargo 60 (PERCENT\_PRI: 1.0, DISTANCE\_KM: 1394.51)  
Funcionário 20 -> Cargo 27 (PERCENT\_PRI: 0.5, DISTANCE\_KM: 2894.83)  
Funcionário 21 -> Cargo 72 (PERCENT\_PRI: 0.95833, DISTANCE\_KM: 1919.11)  
Funcionário 22 -> Cargo 25 (PERCENT\_PRI: 0.97872, DISTANCE\_KM: 1758.24)  
Funcionário 23 -> Cargo 7 (PERCENT\_PRI: 0.94737, DISTANCE\_KM: 3693.04)  
Funcionário 24 -> Cargo 17 (PERCENT\_PRI: 0.46809, DISTANCE\_KM: 765.83)  
Funcionário 25 -> Cargo 18 (PERCENT\_PRI: 0.76596, DISTANCE\_KM: 1137.24)  
Funcionário 26 -> Cargo 73 (PERCENT\_PRI: 0.08511, DISTANCE\_KM: 0.0)  
Funcionário 27 -> Cargo 23 (PERCENT\_PRI: 0.6383, DISTANCE\_KM: 1758.24)  
Funcionário 28 -> Cargo 24 (PERCENT\_PRI: 0.60526, DISTANCE\_KM: 3590.74)  
Funcionário 29 -> Cargo 20 (PERCENT\_PRI: 0.89362, DISTANCE\_KM: 1758.24)  
Funcionário 30 -> Cargo 8 (PERCENT\_PRI: 0.91489, DISTANCE\_KM: 579.73)  
Funcionário 31 -> Cargo 12 (PERCENT\_PRI: 0.80851, DISTANCE\_KM: 801.75)  
Funcionário 32 -> Cargo 59 (PERCENT\_PRI: 0.95745, DISTANCE\_KM: 1394.51)  
Funcionário 33 -> Cargo 62 (PERCENT\_PRI: 0.93617, DISTANCE\_KM: 1113.01)  
Funcionário 34 -> Cargo 26 (PERCENT\_PRI: 0.57447, DISTANCE\_KM: 1831.83)

Funcionário 35 -> Cargo 66 (PERCENT\_PRI: 0.97872, DISTANCE\_KM: 1394.51)  
Funcionário 36 -> Cargo 63 (PERCENT\_PRI: 0.85417, DISTANCE\_KM: 1290.59)  
Funcionário 37 -> Cargo 74 (PERCENT\_PRI: 0.97917, DISTANCE\_KM: 1552.73)  
Funcionário 38 -> Cargo 19 (PERCENT\_PRI: 0.59574, DISTANCE\_KM: 123.06)  
Funcionário 39 -> Cargo 15 (PERCENT\_PRI: 0.93617, DISTANCE\_KM: 1391.76)  
Funcionário 40 -> Cargo 67 (PERCENT\_PRI: 0.91667, DISTANCE\_KM: 1472.82)  
Funcionário 41 -> Cargo 28 (PERCENT\_PRI: 0.6875, DISTANCE\_KM: 1382.29)  
Funcionário 42 -> Cargo 58 (PERCENT\_PRI: 0.22917, DISTANCE\_KM: 1815.15)  
Funcionário 43 -> Cargo 55 (PERCENT\_PRI: 0.87234, DISTANCE\_KM: 333.52)  
Funcionário 44 -> Cargo 14 (PERCENT\_PRI: 0.80851, DISTANCE\_KM: 765.83)  
Funcionário 45 -> Cargo 5 (PERCENT\_PRI: 0.76596, DISTANCE\_KM: 1604.7)  
Funcionário 46 -> Cargo 2 (PERCENT\_PRI: 0.94737, DISTANCE\_KM: 2455.42)  
Funcionário 47 -> Cargo 4 (PERCENT\_PRI: 0.7234, DISTANCE\_KM: 1137.24)  
Funcionário 48 -> Cargo 68 (PERCENT\_PRI: 0.95745, DISTANCE\_KM: 1394.51)  
Funcionário 49 -> Cargo 21 (PERCENT\_PRI: 0.42553, DISTANCE\_KM: 1637.95)  
Funcionário 50 -> Cargo 65 (PERCENT\_PRI: 0.76596, DISTANCE\_KM: 1394.51)  
``