
Alocador Aeroportuário: Otimização de Estacionamentos de Aeronaves

Utilizando Programação por Restrições com OR-
Tools

Visão Geral do Problema

- Aeroportos precisam alocar aeronaves a estacionamentos limitados
- Cada aeronave tem um momento de chegada e tempo de permanência específicos
- Diferentes estacionamentos têm custos variados para cada aeronave
- Objetivo: Minimizar o custo total de alocação respeitando todas as restrições



Restrições do Modelo

- **Exclusividade:** Apenas uma aeronave por estacionamento em cada momento
- **Alocação Única:** Cada aeronave usa no máximo um estacionamento durante toda sua permanência
- **Contiguidade:** A permanência deve ser em um bloco contínuo de tempo
- **Chegada:** A aeronave deve ocupar o estacionamento no momento exato de sua chegada
- **Duração:** A aeronave deve permanecer pelo tempo exato especificado

Modelagem Matemática

Variáveis de Decisão:

- $X[i][j][k] = 1$ se avião i está no estacionamento j no tempo k
- $Y[i][j] = 1$ se avião i usa o estacionamento j em algum momento

Função Objetivo:

Minimizar $\sum(custos[i][j] \times Y[i][j])$

Restrições:

- Implementadas usando o solver CP-SAT do OR-Tools
- Garantem todas as condições operacionais necessárias
- Vinculam as variáveis X e Y para consistência do modelo

Estrutura do Código

- Instalação automática de dependências (pandas, ortools, matplotlib)
- Função principal: `alocar_avioes()` com parâmetros de entrada
- Criação do modelo com variáveis e restrições
- Resolução do modelo usando CP-SAT solver
- Visualização dos resultados com matplotlib

```
def alocar_avioes(num_avioes, num_estacionamentos,
                  horizonte, chegadas, duracoes, custos):

    # Criar modelo
    model = cp_model.CpModel()

    # Definir variáveis
    x = {}  # x[i][j][k] = 1 se avião i está no estacionamento j no tempo k
    y = {}  # y[i][j] = 1 se avião i usa o estacionamento j

    # Adicionar restrições

    # Resolver o modelo
    solver = cp_model.CpSolver()
    status = solver.Solve(model)

    # Retornar e visualizar resultados
```

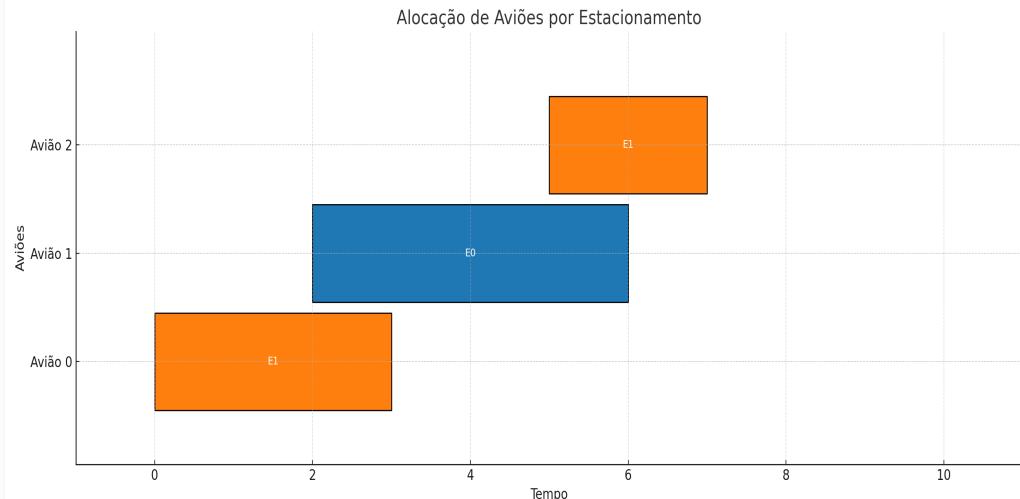
Exemplo Pequeno: Dados de Entrada

Parâmetro	Valor
Número de aviões	3
Número de estacionamentos	2
Horizonte de tempo	10 unidades
Chegadas	[0, 2, 5]
Durações	[3, 4, 2]
Matriz de Custos	Avião 0: [10, 50] Avião 1: [10, 60] Avião 2: [40, 5]

Exemplo Pequeno: Resultados

Solução Ótima

- **Custo total mínimo: 65.0**
- Alocações:
 - Avião 0 → Estacionamento 1
(tempo 0-2, custo 50)
 - Avião 1 → Estacionamento 0
(tempo 2-5, custo 10)
 - Avião 2 → Estacionamento 1
(tempo 5-6, custo 5)



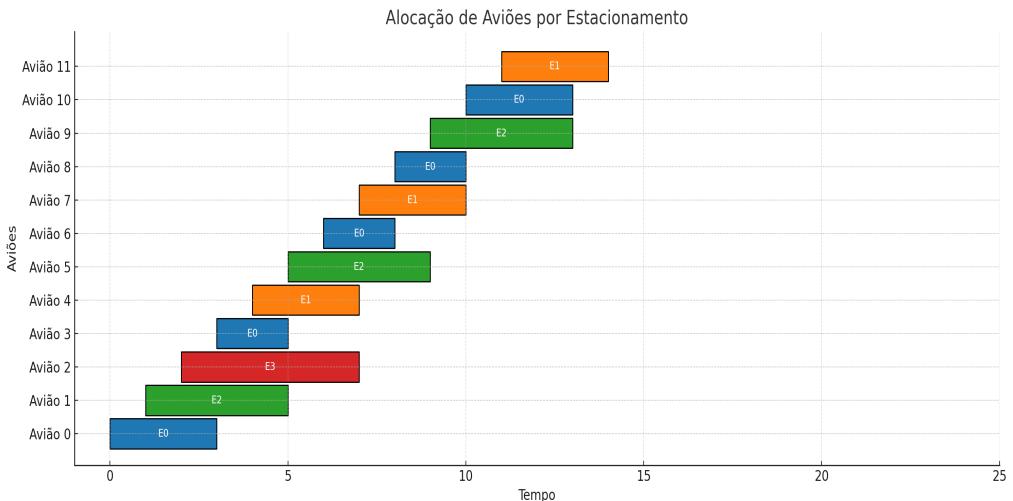
Exemplo Grande: Dados de Entrada

Parâmetro	Valor
Número de aviões	12
Número de estacionamentos	5
Horizonte de tempo	24 unidades
Chegadas	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Durações	[3, 4, 5, 2, 3, 4, 2, 3, 2, 4, 3, 3]
Matriz de Custos	Custos crescentes com o índice do estacionamento

Exemplo Grande: Resultados

Solução Ótima

- **Custo total mínimo: 2730.0**
- Todas as 12 aeronaves alocadas com sucesso
- Distribuição entre os 5 estacionamentos disponíveis
- Utilização eficiente do espaço temporal
- Tempo de resolução: 0.42 segundos



Visualização da Solução

- Gráfico de Gantt mostra a alocação temporal de cada avião
- Eixo Y: Aviões
- Eixo X: Unidades de tempo
- Cores diferentes para cada estacionamento
- Rótulos indicam qual estacionamento foi utilizado

```
def visualizar_resultado(num_avioes, horizonte,
                         resultado, estacionamentos):
    plt.figure(figsize=(12, 6))
    cores = ['#3498db', '#2ecc71', '#e74c3c',
             '#f39c12', '#9b59b6']

    for i in range(num_avioes):
        for j, alocacao in enumerate(resultado[i]):
            if alocacao:
                plt.barh(i, alocacao[1]-alocacao[0],
                         left=alocacao[0],
                         color=cores[estacionamentos[i][j]])

    plt.xlabel('Tempo')
    plt.ylabel('Avião')
    plt.grid(axis='x')
    plt.tight_layout()
    plt.show()
```

Desafios e Soluções

Desafios

- Dependências ausentes (pandas)
- Modelo inicialmente inviável
(INFEASIBLE)
- Visualização clara dos resultados
- Escalabilidade para problemas grandes

Soluções

- Instalação automática de pacotes
- Refinamento das restrições de contiguidade e vinculação X-Y
- Gráfico de Gantt personalizado com cores por estacionamento
- Otimização do modelo com propagação eficiente de restrições

Aplicações Práticas

- Planejamento operacional de aeroportos
- Otimização de recursos em terminais
- Redução de custos operacionais
- Adaptável para outros problemas de alocação de recursos:
 - Docas em portos
 - Salas em hospitais
 - Máquinas em fábricas



Conclusões

- Programação por restrições é eficaz para problemas de alocação complexos
- OR-Tools oferece um framework poderoso para modelagem e resolução
- Visualização adequada facilita a interpretação dos resultados
- Modelo pode ser estendido para incluir mais restrições operacionais
- Potencial para economia significativa em operações aeroportuárias

Próximos Passos

- Incorporar restrições adicionais (tipos de aeronaves, equipamentos)
- Implementar interface gráfica para usuários
- Integrar com sistemas de gerenciamento aeroportuário
- Adicionar otimização multi-objetivo (custo vs. conveniência)
- Desenvolver versão em tempo real para ajustes dinâmicos