

# Orientações para Entrega do Código

## ELE634 – Laboratório de Sistemas II

André Costa Batista

Universidade Federal de Minas Gerais

24 de outubro de 2025

# Sumário

- 1 Introdução
- 2 Critério de Parada
- 3 Estrutura da Entrega
- 4 Método resolva
- 5 Estrutura da Classe Solucao
- 6 Número de Avaliações por Instância
- 7 Validação e Avaliação
- 8 Recomendações

## Objetivo

Estabelecer o padrão para entrega das implementações dos algoritmos de otimização

- Garantir execução padronizada de todas as implementações
- Permitir comparação justa entre diferentes abordagens
- Facilitar a avaliação automatizada

## Importante

Todas as especificações devem ser seguidas!

## Número de Avaliações da Função Objetivo

Critério único para todos os algoritmos

### O que conta como uma avaliação?

- Cada chamada à função que calcula o valor da função objetivo
- Deve ser contabilizada explicitamente no código

### Quando parar?

- Ao atingir o número máximo de avaliações especificado
- Retornar a **melhor solução encontrada** até então

### E quem for usar o Gurobi como busca local?

- Controlar os seguintes parâmetros do Gurobi: `NodeLimit` e `IterationLimit`
- Somar ao número corrente de avaliações após a execução do Gurobi os seguintes valores: `NodeCount` e `IterCount`

## Verificação

O código será revisado para verificar o correto controle de avaliações!

# Formato do Arquivo

- 1 Pasta **zipada** com nome do grupo
- 2 Arquivo Python principal com **mesmo nome do grupo**
- 3 Método resolva no arquivo principal
- 4 Arquivos auxiliares opcionais

## Exemplo: Grupo “reginaldorossi”

```
reginaldorossi.zip
|-- reginaldorossi.py      # Principal (obrigatorio)
|-- heurísticas.py        # Auxiliar (opcional)
|-- operadores.py         # Auxiliar (opcional)
'-- utils.py              # Auxiliar (opcional)
```

# Assinatura da Função

```
1 def resolve(dados: Dados, numero_avaliacoes: int) -> Solucao:
2     """
3     Executa o algoritmo de otimizacao.
4
5     Parametros:
6     -----
7     dados : Dados
8         Objeto com os dados da instancia
9
10    numero_avaliacoes : int
11        Numero maximo de avaliacoes permitidas
12
13    Retorna:
14    -----
15    Solucao
16        Objeto com rotas, tempos e funcao objetivo
17    """
18    # Implementacao do algoritmo aqui
19    pass
20
```

`dados` Objeto da classe Dados

- Mesma classe do arquivo `dados.py` do repositório
- Contém todos os parâmetros da instância

`numero_avaliacoes` Inteiro definindo o critério de parada

- Algoritmo deve parar ao atingir este limite
- Controle rigoroso é obrigatório

## Objeto da Classe Solucao

Mesma classe do arquivo `solucao.py` do repositório

### Deve conter:

- **Rotas** (rota): Sequência de requisições por ônibus/viagem
- **Tempos de chegada** (chegada): Instantes de chegada em cada ponto
- **Função objetivo** (fx): Valor calculado da FO



# Atributos da Classe Solucao

`rota[k][v]` Lista de requisições visitadas

- Ônibus  $k$  na viagem  $v$
- Sempre inicia e termina com 0 (garagem)

`chegada[k][v]` Lista de tempos (float)

- Instantes de chegada correspondentes à rota
- Mesmo tamanho de `rota[k][v]`
- Primeiro elemento sempre 0.0

`fx` Valor da função objetivo (float)

# Exemplo de Configuração (1/2)

```
1 from solucao import Solucao
2
3 # Criar objeto de solucao
4 solucao = Solucao()
5
6 # Configurar rotas
7 # Onibus 1 realiza 3 viagens
8 solucao.rota[1] = {}
9 solucao.rota[1][1] = [0, 2, 4, 0] # Viagem 1: garagem -> req 2 -> req 4 -> garagem
10 solucao.rota[1][2] = [0, 6, 8, 11, 0] # Viagem 2: garagem -> req 6 -> req 8 -> req 11 -> garagem
11 solucao.rota[1][3] = [0, 13, 0] # Viagem 3: garagem -> req 13 -> garagem
12 solucao.rota[1][4] = [] # Viagem 4: nao utilizada
13
```

## Exemplo de Configuração (2/2)

```
1 # Tempos correspondentes as rotas
2 solucao.chegada[1] = {}
3 solucao.chegada[1][1] = [3.89, 17.0, 31.08, 80.2]      # Tempos para rota[1][1]
4 solucao.chegada[1][2] = [80.2, 92.0, 132.5, 185.0, 200.2] # Tempos para rota[1][2]
5 solucao.chegada[1][3] = [202.12, 215.0, 223.0]         # Tempos para rota[1][3]
6 solucao.chegada[1][4] = []                             # Viagem nao utilizada
7
8 # Funcao objetivo
9 solucao.fx = 33486.4
10
```

## Atenção!

- **Índice 0 é a garagem:** Toda rota inicia e termina com 0
- **Correspondência exata:** `len(rota[k][v]) == len(chegada[k][v])`
- **Viagens não utilizadas:** Listas vazias `[]`
- **Saída da garagem (primeira chegada):** : Instante em que o ônibus começa o preparo para a viagem. Ou seja, é igual ao instante em que o ônibus chega na primeira requisição da viagem menos o tempo de deslocamento da garagem até essa requisição e menos o tempo de preparo inicial (que o tempo de serviço relativo à garagem).
- **Chegada na garagem:** Instante em que o ônibus retorna à garagem após a última requisição da viagem.
- **Todos ônibus/viagens:** Devem ter entradas no dicionário
- **IMPORTANTE:** Dentro da execução do seu algoritmo, você pode utilizar a estrutura de dados que quiser. A estrutura apresentada aqui é apenas para a **saída final** do método `resolva` através do objeto `Solucao`.

## Número Máximo de Avaliações

$$N_{av}^{max} = 10 \times n \times K \times r$$

### Onde:

- $n$  = número de requisições
- $K$  = número de ônibus disponíveis
- $r$  = número máximo de viagens por ônibus

# Valores por Instância

Instância	n	K	r	$N_{av}^{max}$
pequena	14	3	5	2.100
média	67	6	12	48.240
grande	108	11	10	118.800
rush	108	11	10	118.800

## Importante

É com esses valores que os algoritmos serão comparados. Nos testes de vocês, podem usar os valores que quiserem.

## Verificações Automáticas:

### ① **Viabilidade da solução:** Todas as restrições respeitadas

- Todas requisições atendidas
- Janelas de tempo
- Número máximo de viagens
- Duração total das viagens

### ② **Cálculo da função objetivo**

- Conferência do valor reportado
- Verificação de consistência

## Atenção

Soluções **inviáveis** serão descartadas da amostra para comparação!

## Cada Algoritmo

- Executado **30 vezes** em cada instância
- Cada chamada de `resolva` = 1 execução
- Total: 30 execuções  $\times$  4 instâncias = 120 execuções

## Análise Estatística:

- Média (ou mediana) dos valores da FO
- Apenas soluções viáveis consideradas
- Gráficos de boxplot para comparação



- **Controle rigoroso:** Certifique-se de respeitar o limite de avaliações
- **Tratamento de erros:** Evite crashes com exceções adequadas
- **Documentação:** Comente o código, especialmente `resolve`
- **Testes locais:** Valide com as instâncias fornecidas

# Dúvidas?