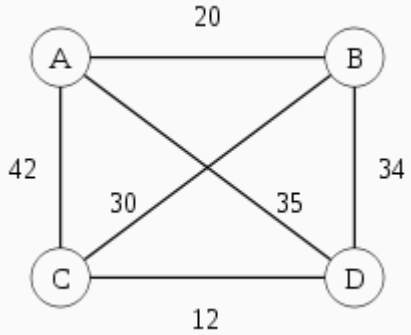
# Resolução do Problema do Mochileiro Viajante via Algoritmo Genético + Busca Local 2-Opt

Inteligência Computacional

Felipe R2
João Marcos

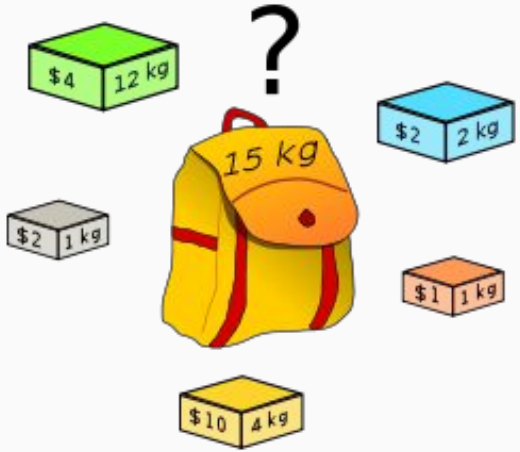
# Mochileio Viajante

- Junção de dois problemas clássicos: Caixeiro viajante e problema da mochilha.
- Caixeiro viajante: É ilustrado como uma rota que passa por todos os pontos de um conjunto determinado onde se deve garantir que todos os pontos foram visitados. No entanto, o caminho entre 2 pontos do conjunto possui um certo custo, que atribui uma medida de qualidade. Logo, o objetivo é encontrar a rota que tenha o melhor custo dado problema abordado.



# Mochileio Viajante

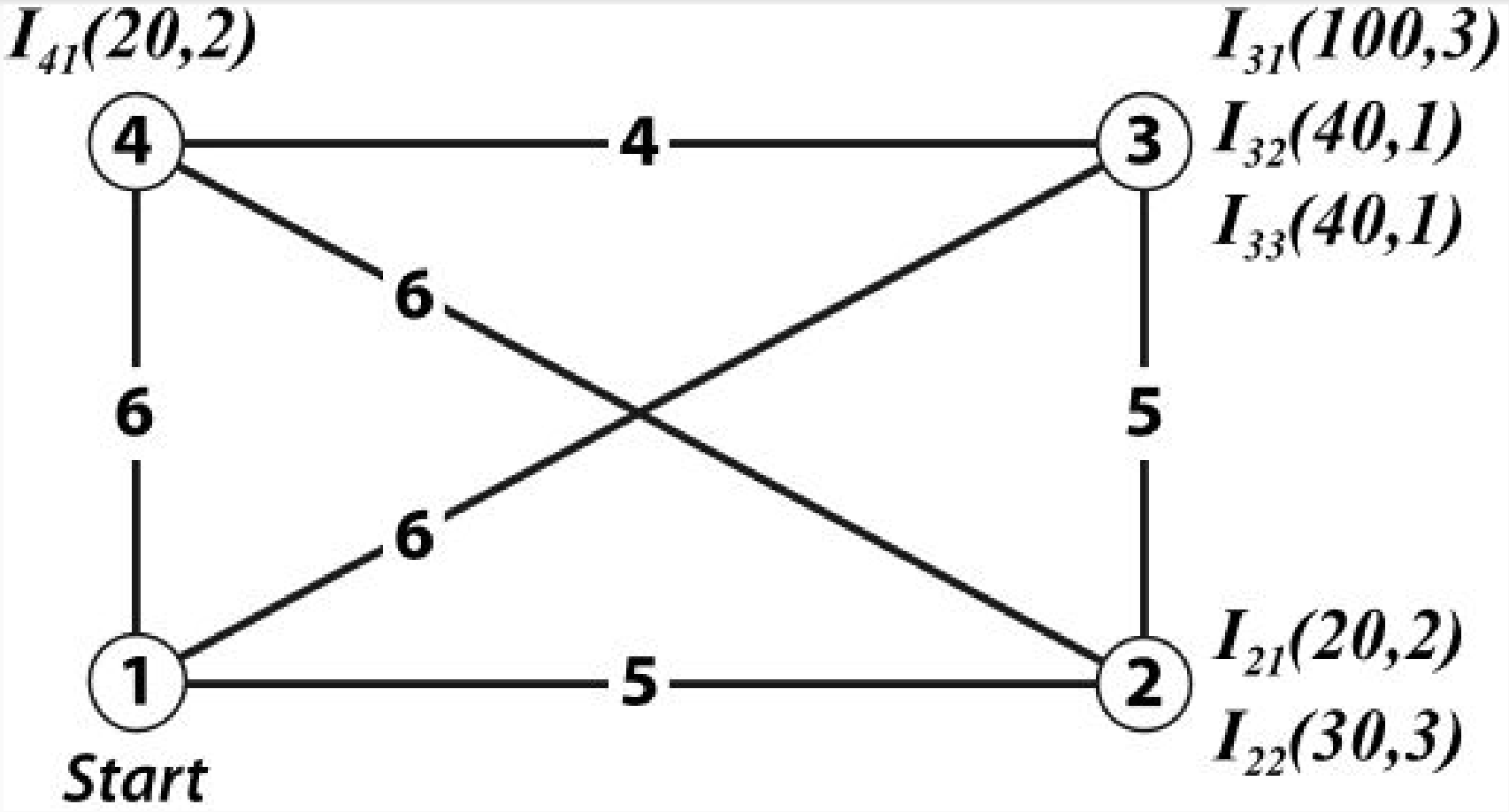
- Problema da Mochila: É ilustrado como inserir em uma mochila um número determinado de itens de forma que não viole alguma restrição, como o espaço da mochila ou peso máximo que ela pode carregar. No entanto, cada item possui um valor atribuído, então o objetivo passa a ser encontrar a melhor combinação de itens que maximize este valor.



# Mochileio Viajante

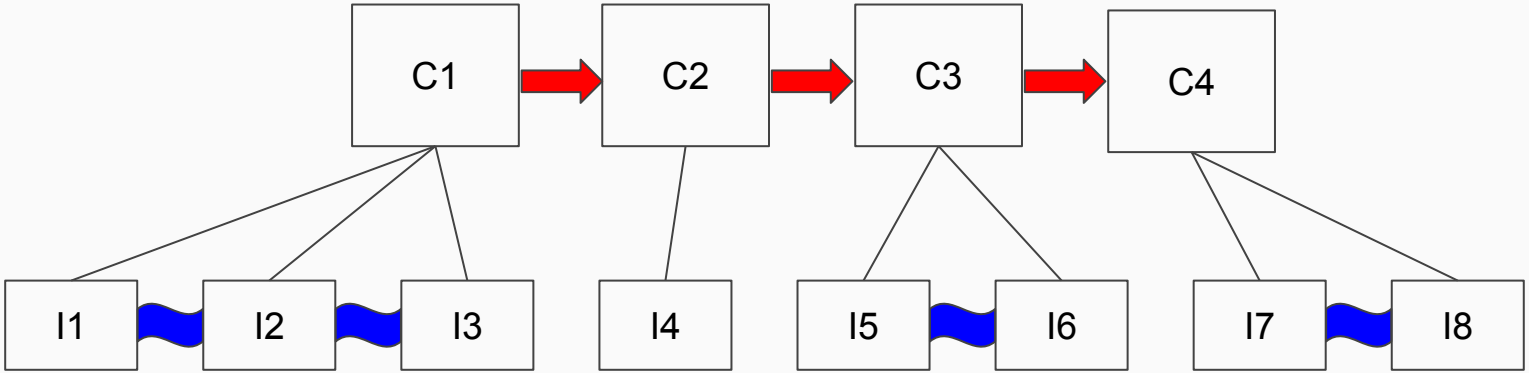
- O Mochileiro Viajante assume o papel do caixeiro que precisa determinar uma rota de menor custo, e, ao mesmo tempo, encher uma mochila de forma a maximizar seu ganho com os itens selecionados.
- Cada ponto possui um conjunto de itens.
- Cada item possui um peso e valor.
- O tempo de viagem entre dois pontos depende de sua distância e do peso da mochila nesse trajeto.
- Quanto mais pesada a mochila mais lentamente o mochileiro viaja.
- Ao final, é descontado do ganho da mochila (G) o custo da viagem (C) que é o tempo de viagem vezes o aluguel da mochila.

# Mochileio Viajante



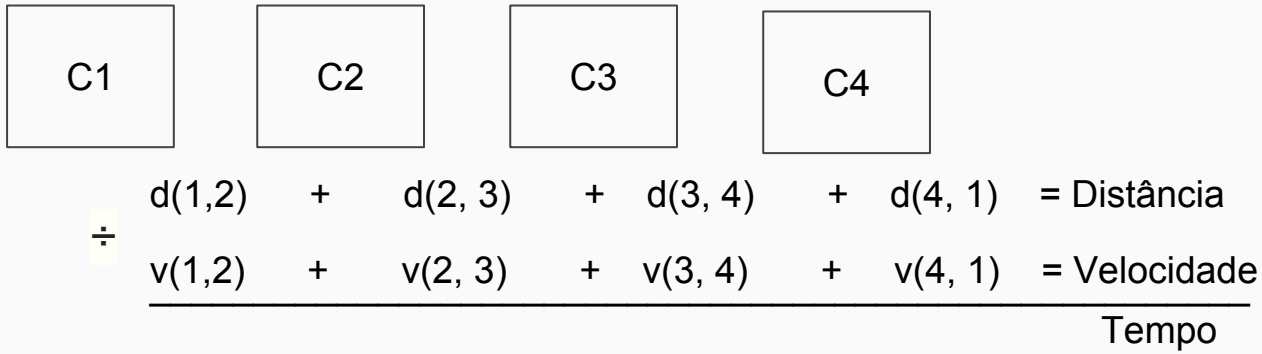
# Representação da Solução

- Uma solução do problema deve conter:
  - Uma configuração de rota.
  - Uma configuração de mochila.



# Representação da Solução

- Alguns valores devem ser facilmente calculados:
  - Tempo, distância(i, j) e velocidade(i, j),
  - Peso(i, j), valor(i, j).



# Representação da Solução

- Alguns valores devem ser facilmente calculados:
  - Tempo, distância (i, j) e velocidade(i, j),
  - Peso(i, j), valor(i, j).

| I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 |
|---|---|---|---|---|---|---|---|

p(1) + p(2) + p(3) + p(4) + p(5) + p(6) + p(7) + p(8) = Peso

v(1) + v(2) + v(3) + v(4) + v(5) + v(6) + v(7) + v(8) = Valor

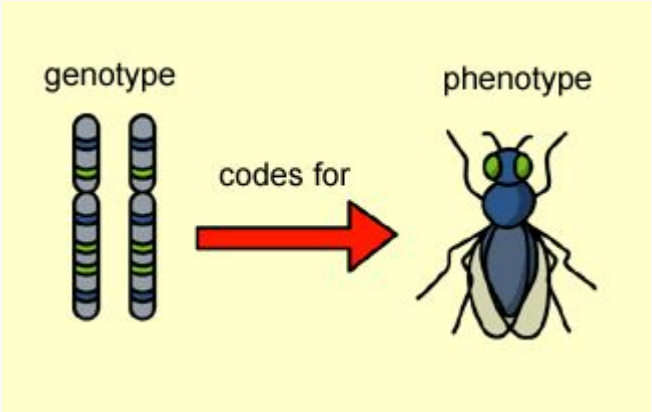
# Algoritmo Genético

- Inspirado na teoria da Evolução Natural proposta por Darwin o AG se baseia em evoluir uma população de indivíduos de forma similar à vista na natureza.

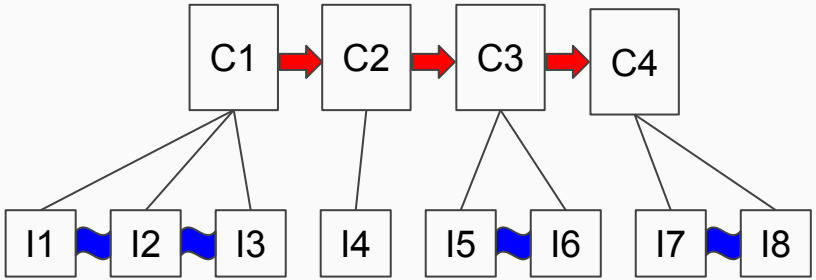
- Uma solução proposta para um problema é codificada como o “DNA”, ou genótipo, de um indivíduo da “espécie” solução. A expressão do genótipo é a solução em si, ou, fenótipo.

- A imagem a seguir ilustra um paralelo entre a representação natural e a artificial.

# Algoritmo Genético




genótipo




fenótipo

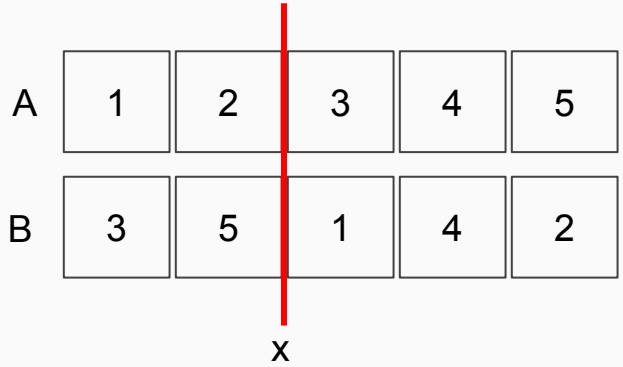
Lucro

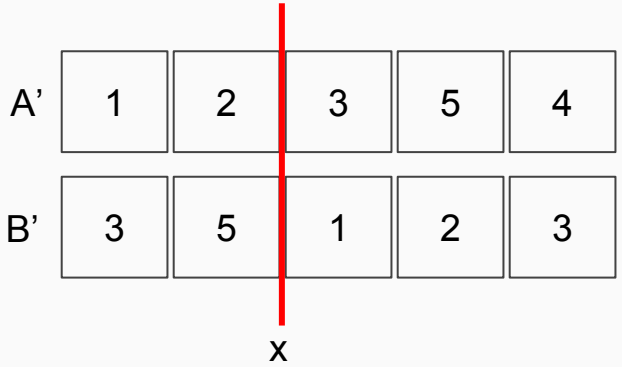
# Algoritmo Genético

- Componentes do AG
  - População: conjunto de agentes de uma determinada espécie, soluções candidatas.
  - Operadores:
    - Recombinação: simula aqui a reprodução sexuada de vertebrados, onde material genético de um casal é recombinado em sua prole.
    - Mutação: provoca uma perturbação no genótipo, introduzindo novas informações à população.
    - Seleções: utilizadas para decidir (i) quem participa na reprodução e (ii) quem, entre uma população antiga e uma nova ficará para a próxima geração.

# Algoritmo Genético

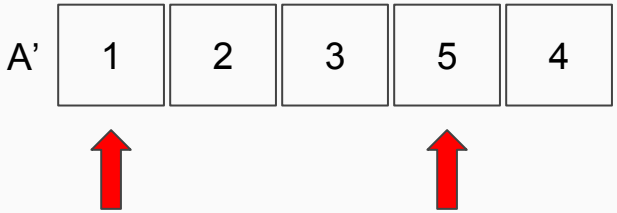
- Recombinação
  - Seleciona 2 indivíduos bem adaptados, A e B
  - Seleciona um ponto x em [1, n], onde n é o número de cidades.
  - Cria A’ com as cidades de A até x, assim como B’ com as cidades de B.
  - Completa as cidade de x+1 até n em A’ com as cidades de B que ainda não estão em A’, e o mesmo com B’ e A.

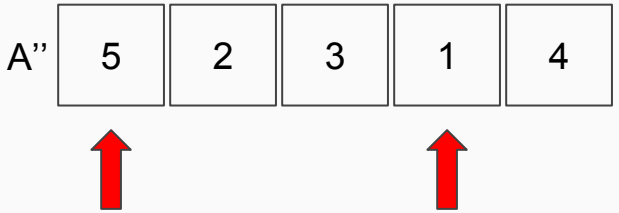


# Algoritmo Genético

- Mutação
  - Seleciona 2 cidades em um indivíduo.
  - Troca essas cidades.
  - Gera um novo indivíduo A'', mas distante apenas 1 movimento de A'.



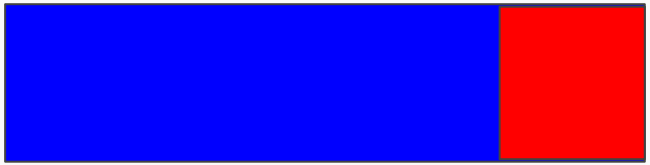



# Algoritmo Genético

- Seleção
  - Torneio:
    - Competição feita entre 2 indivíduos A e B.
    - Vence qual é mais adaptado.
  - Substituição da População:
    - Duas populações competem para permanecer na próxima geração.
    - É utilizado o elitismo, onde a elite (vermelho) de ambas populações permanecem, em proporções diferentes.

População Corrente

População Recém Criada



# Algoritmo Construtivo

- Base:
  - Criar uma rota viável.
  - Preencher a mochila.

- Para a rota:
  - Aleatório:
    - Um vetor com todas as cidades é embaralhado.
  - Heurístico Guloso Randomizado:
    - Para cada par (cidade corrente, próxima cidade) é calculado o movimento mais vantajoso.
    - Randomizado pois pode aceitar os 60% movimentos mais vantajosos.

# Algoritmo Construtivo

- Base:
  - Criar uma rota viável.
  - Preencher a mochila.

- Para a mochila:
  - Ordenado:
    - Os itens em ordem das cidades são inseridos até que a mochila não aceite mais itens.
  - Heurístico Guloso Randomizado:
    - Os itens são ordenados pelo seu custo benefício e então são inseridos nessa ordem até que a mochila não aceite mais itens.

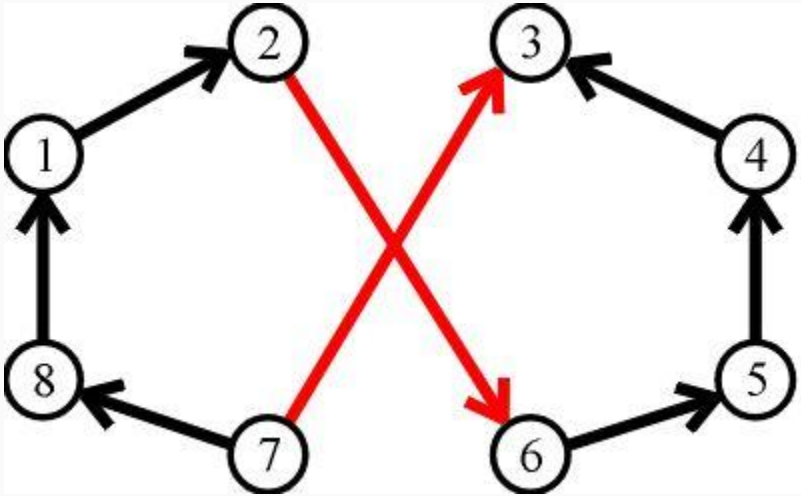
# Algoritmo Construtivo

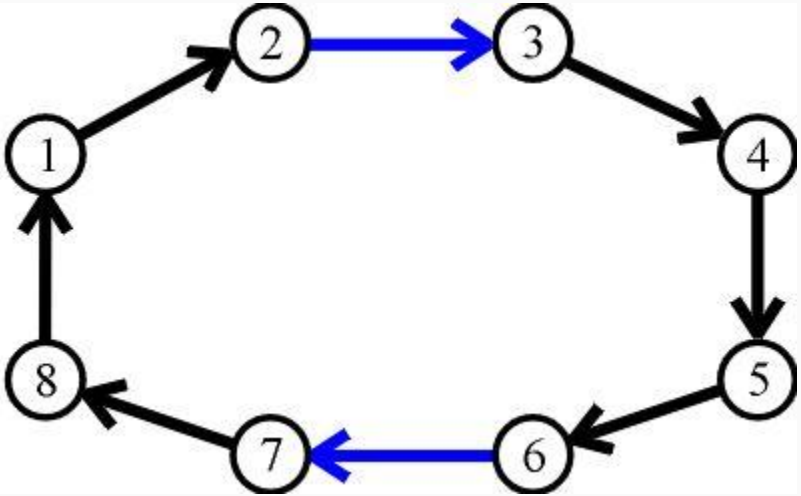
- Base:
  - Criar uma rota viável.
  - Preencher a mochila.

- Para a rota:
  - 50% de chance para cada abordagem.
  - Permite grande variedade nas soluções iniciais e ainda assim inserir algum conhecimento heurístico.
- Para a mochila:
  - Necessário rebuscar melhor a mochila.

# Busca Local

- A Busca Local aqui proposta é baseada em definir a melhor rota possível para uma mochila específica.
- Busca 2-opt é utilizada para melhorar uma solução.
- É baseado em remover 2 arestas do grafo que representa a rota e realizar um rearranjo.



# Algoritmos

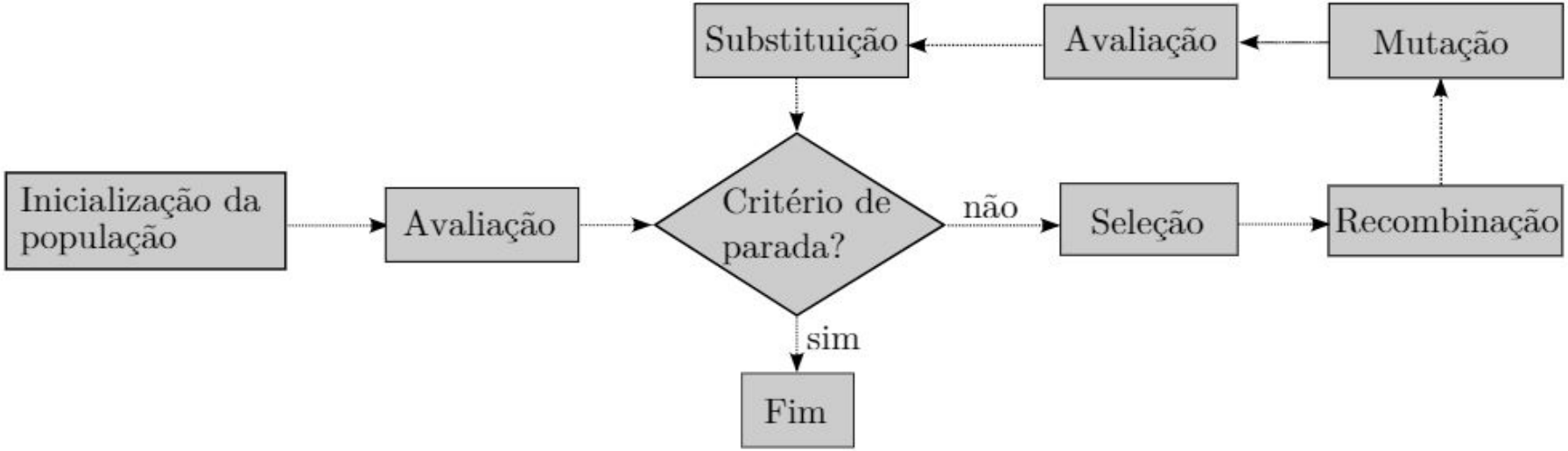



Figura 2.1: Fluxograma AG genérico.

# Algoritmos

```
Rota 2-Opt(Rota rota[size]){
    int size = n cidades;
    int melhora = 0;

    while (melhora < alpha){
        double minDistancia = Distancia(cidades);
        for ( int i = 0; i < size - 1; i++ ){
            for ( int k = i + 1; k < size; k++){
                novaRota[] = cidades;
                Rearranjar(rota, novaRota, i, k);
                double novaDistancia = Distancia(novaRota);

                if (novaDistancia < minDistancia)
                    melhora++;
                    swap(rota, novaRota);
                    minDistancia = novaDistancia;
                }
            }
        }
    }
    return rota;
}
```

# Algoritmo Genético com Busca Local 2-Opt

- Cria população
- Enquanto critério não atende
    - Seleção
    - Gera nova população recombinando
    - Aplica mutação
    - Aplica Busca Local
    - Elitismo

# Experimentos Computacionais

- AG
  - População de 5000 indivíduos
  - Mutação ocorre 100% das vezes
  - Critério de parada: tempo
  - Elitismo de 20%
- 2-Opt
  - Realiza no máximo 3 iterações ou para quando ocorre uma melhora

# Resultados

- Em experimentos preliminares utilizamos a classe eil51-ttp com 50 cidades e 50 itens
    - 01: L =  4343,57 e média  4301,15
    - 02: L =  5216,92 e média  5066,17
    - 05: L = -2006,68 e média -2249,21

# Resultados

- Funcionou bem, no entanto, é necessário um ajuste mais sistemático de parâmetros.
- Criar operações focadas na mochila e não tão fortemente na rota.
- Executar para mais instâncias.