

Problema do Caixeiro-Viajante

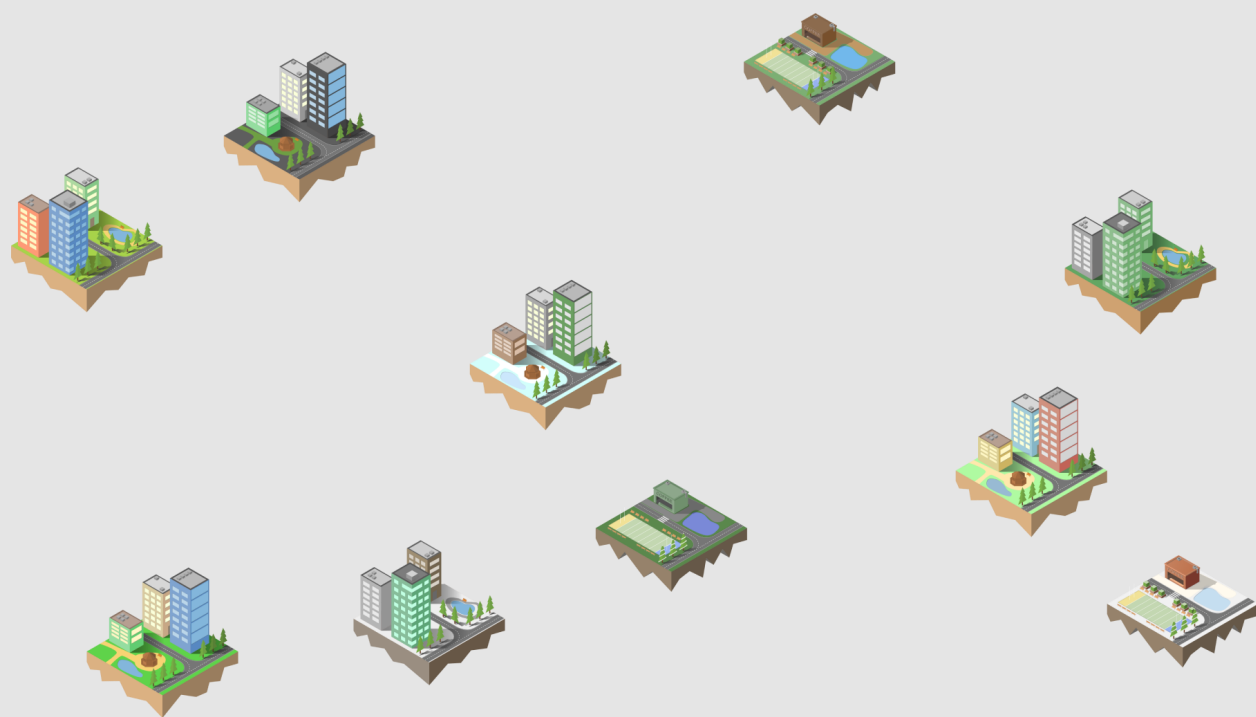
Algoritmo Genético Multi-Objetivo

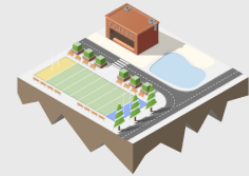
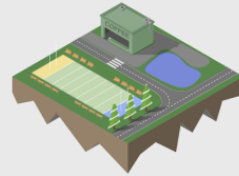
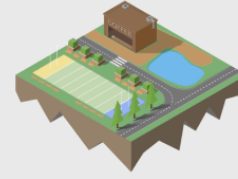


Matheus Lôbo dos Santos

O que é o Problema do Caixeiro-Viajante?

O Problema do Caixeiro Viajante (TSP, da sigla em inglês) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem.





Complexidade do Problema

Para 10 cidades:
 $10! = 3.628.800$

Para 20 cidades:
 $20! = 2.4 \times 10^{18}$



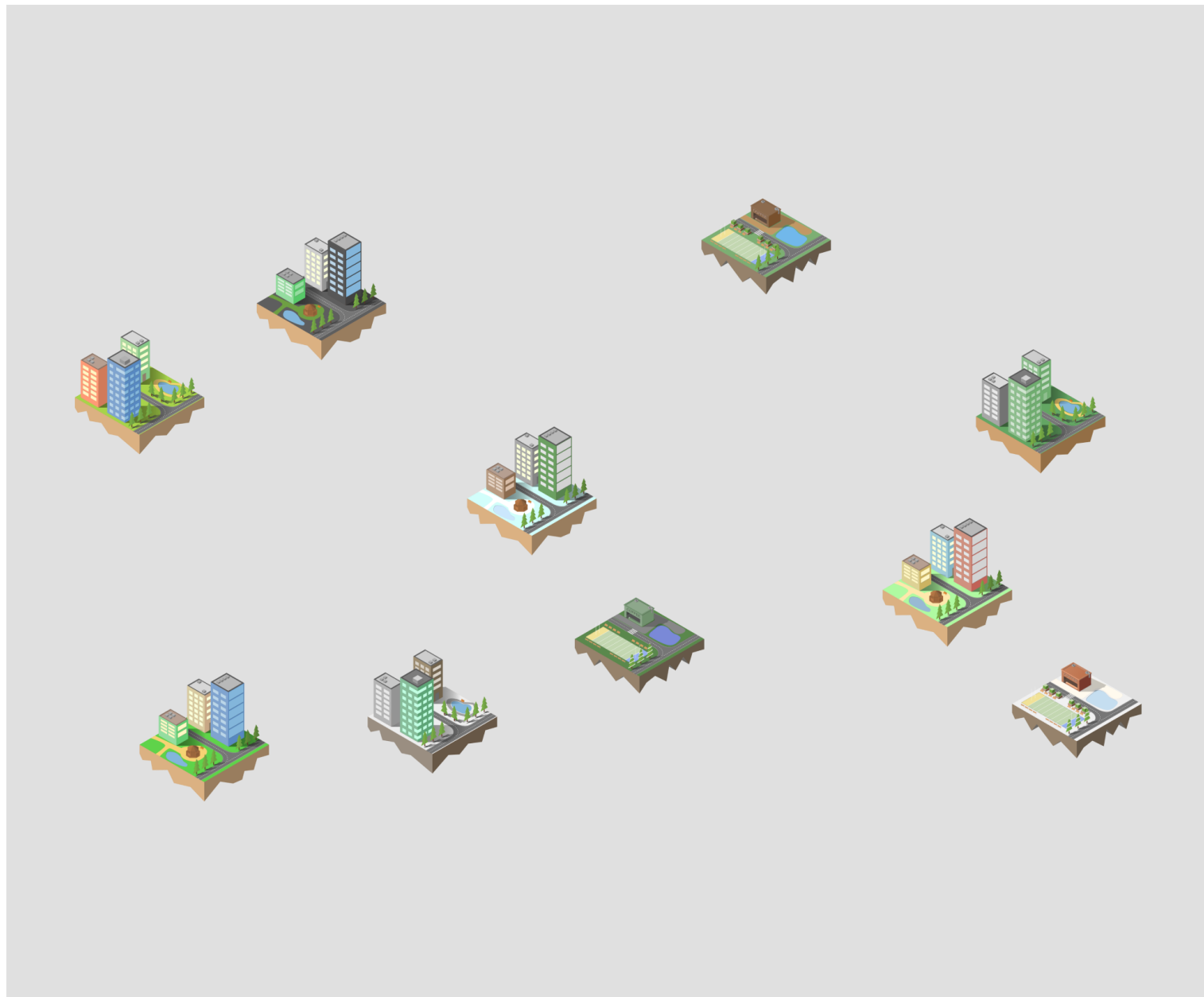
- Problema de otimização NP-difícil;
- O número de soluções disponíveis é dado pelo fatorial do número de cidades.

Algortimo Genético

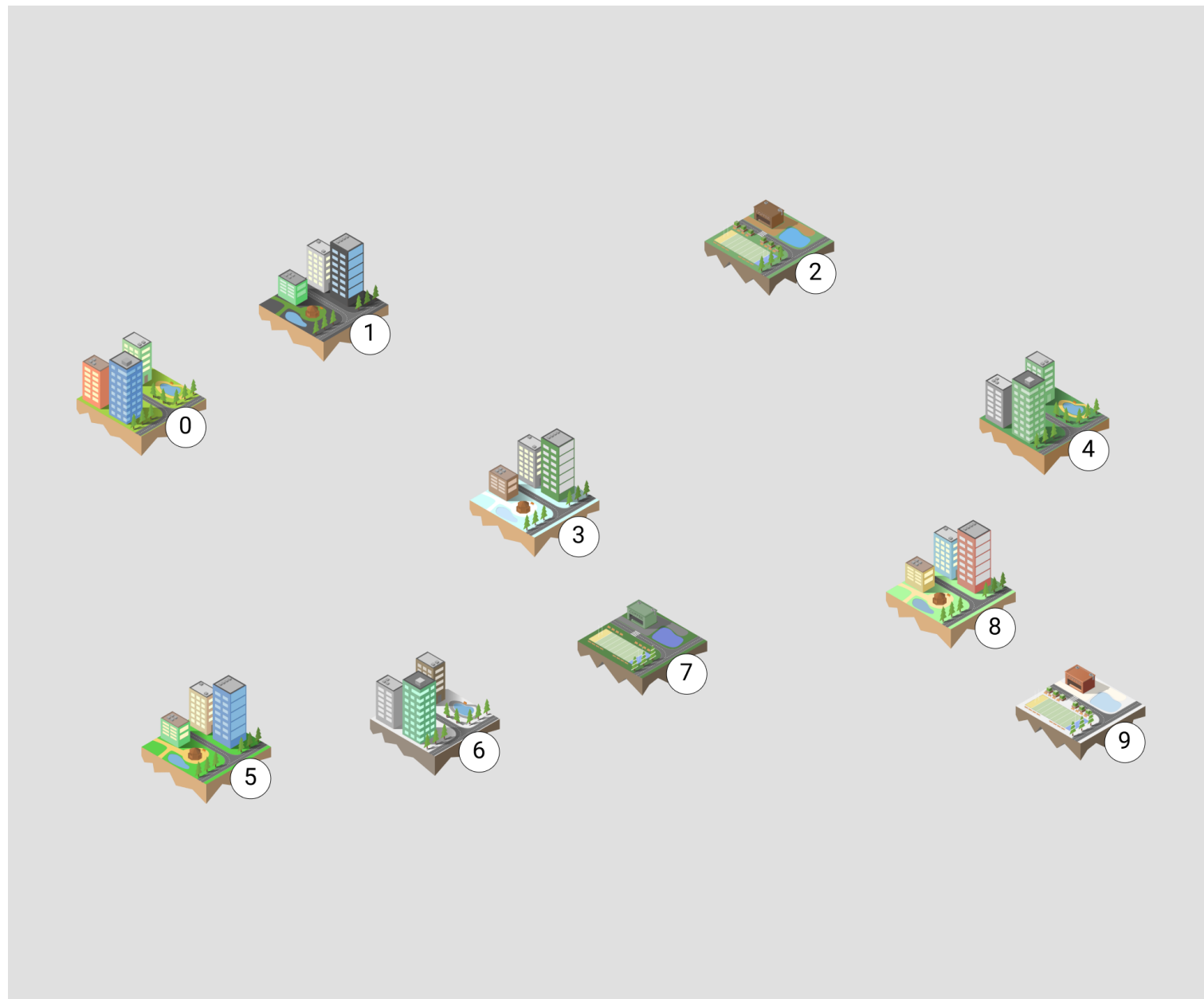
Componentes necessários:

1. Geração (Cria a população);
2. Avaliação (Avalia o desempenho do indivíduo);
3. Cruzamento (Criar novos indivíduos);
4. Mutação (Altera os genes dos indivíduos);

Como Representar um Indivíduo?



Como Representar um Indivíduo?



Como Representar um Indivíduo?

- Indivíduo 1 = [1, 3, 7, 4, 0, 2, 9, 8, 5, 6]
- Indivíduo 2 = [6, 3, 7, 9, 5, 1, 8, 2, 0, 4]

Código | Criação da População

```
public static List<Individual> SpawnPopulation(){
    List<Individual> population = new ArrayList<Individual>();

    // Generate {PopulationCount} individuals
    while (population.size() < Config.populationCounts)
    {
        Individual individual = GenerateIndividual(Config.numberOfCities);
        if (!population.contains(individual))
        {
            population.add(individual);
        }
    }

    return population;
}
```


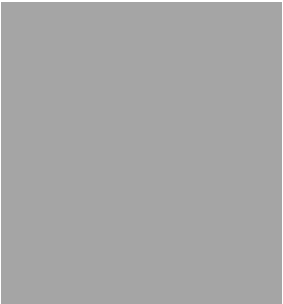
```
public static Individual GenerateIndividual(int sequenceLength){
    List<Integer> sequence = new ArrayList<Integer>();

    for (int i = 0; i < Config.numberOfCities; i++){
        sequence.add(i);
    }

    Utility.randomize(sequence);

    return new Individual(sequence);
}
```

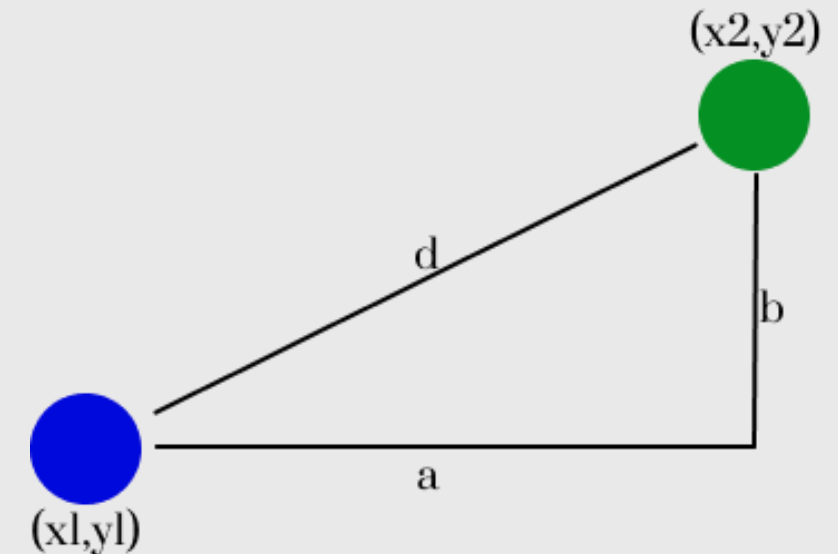
Métricas para a Avaliação

- 
- 
- Distância total da viagem;
 - Tempo total de viagem.

Distância da Viagem

```
private double GetTotalDistance() {  
    double totalDistance = 0.0;  
  
    for (int i = 0; i < sequence.size(); i++){  
        Vector2f fromCitie = TownHelper.townPositions.get(sequence.get(i));  
        Vector2f toCitie = TownHelper.townPositions.get(sequence.get((i + 1)%Config.numberOfCities));  
  
        double x = fromCitie.X - toCitie.X;  
        double y = fromCitie.Y - toCitie.Y;  
  
        double d = Math.sqrt(x * x + y * y);  
  
        totalDistance += d;  
    }  
  
    return totalDistance;  
}
```

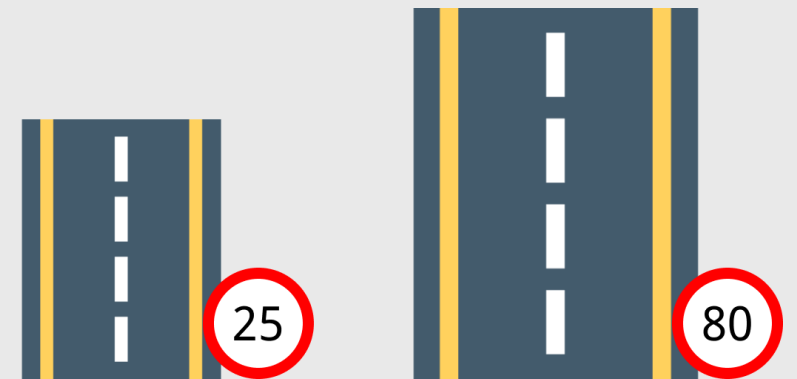
- Cada cidade é representada por uma coordenada x e y;
- A distância entre duas cidades é dada pela distância Euclidiana;
- A soma das distâncias entre as cidades representa a distância final da viagem.

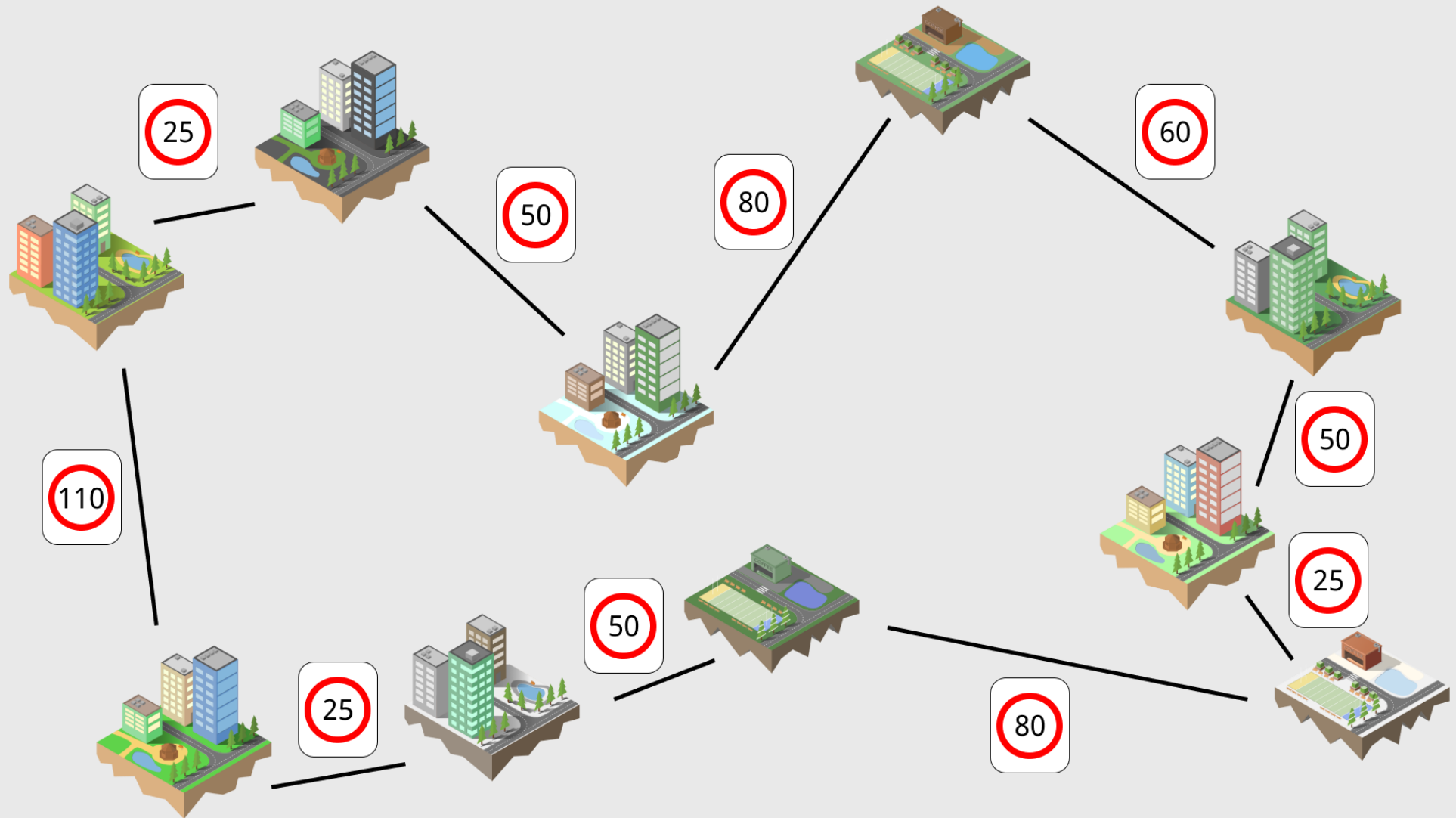


Tempo de Viagem

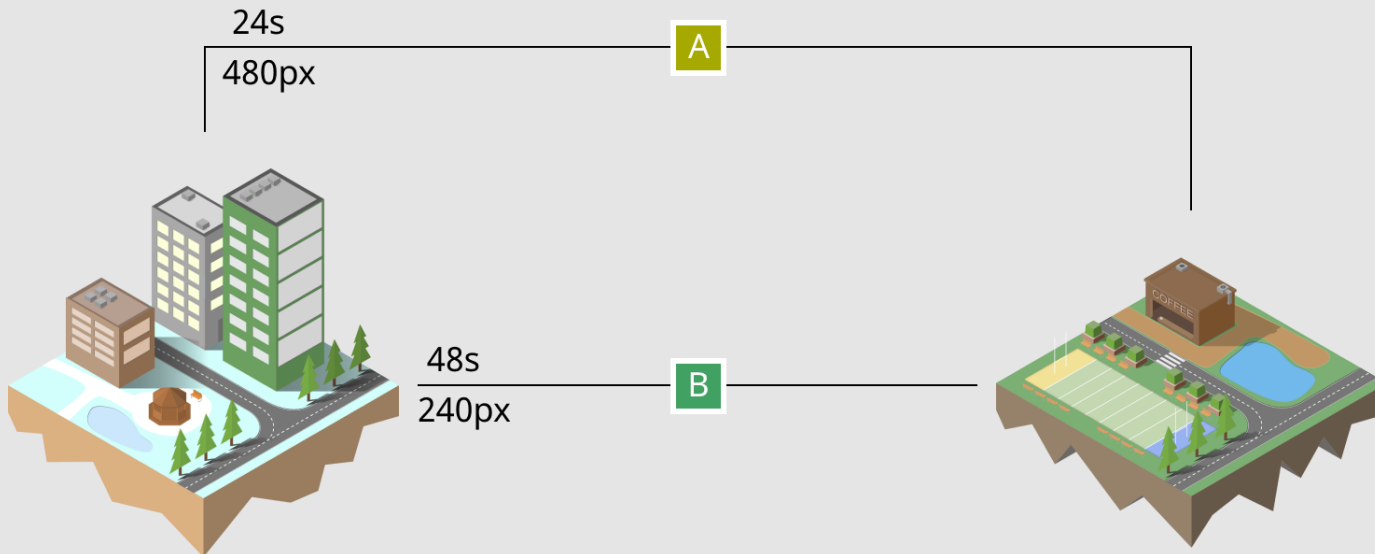
```
private double GetTotalTime() {  
    double totalTime = 0.0;  
  
    for (int i = 0; i < sequence.size(); i++){  
        Vector2f fromCitie = TownHelper.townPositions.get(sequence.get(i));  
        Vector2f toCitie = TownHelper.townPositions.get(sequence.get((i + 1)%Config.numberOfCities));  
  
        double x = fromCitie.X - toCitie.X;  
        double y = fromCitie.Y - toCitie.Y;  
  
        double d = Math.sqrt(x * x + y * y);  
  
        totalTime += d / TownHelper.pathSpeedLimits[sequence.get(i)][sequence.get((i + 1)%Config.numberOfCities)];  
    }  
  
    return totalTime;  
}
```

- Cada rota apresenta uma velocidade máxima;
- A matriz de velocidade é impactada pelo comprimento da rota. Rotas maiores, tendem a apresentar velocidades maiores.
- A soma dos tempos entre as cidades representa o tempo final da viagem.

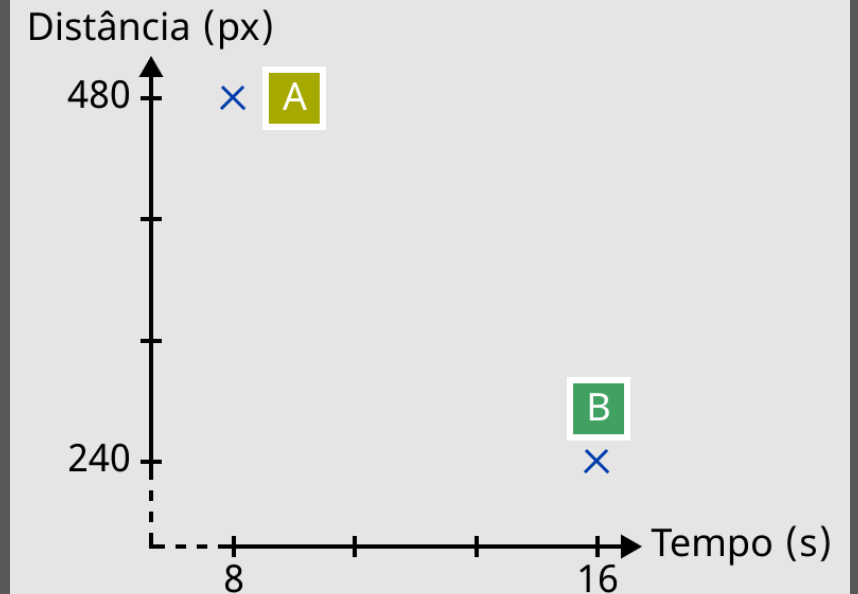




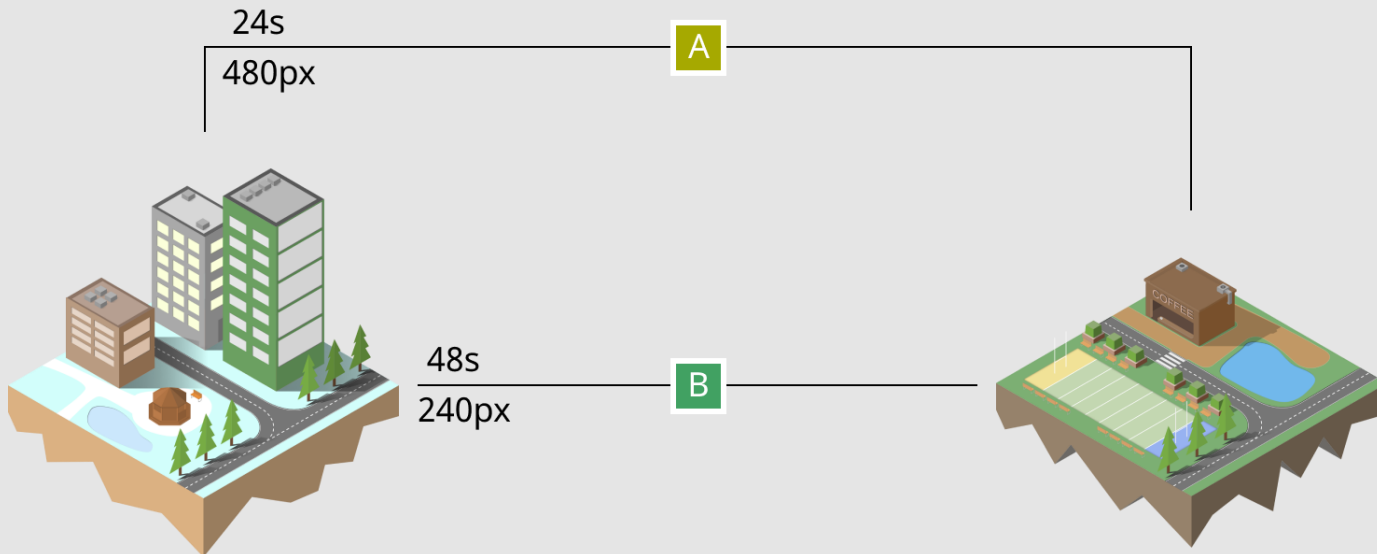
Como combinar as métricas?



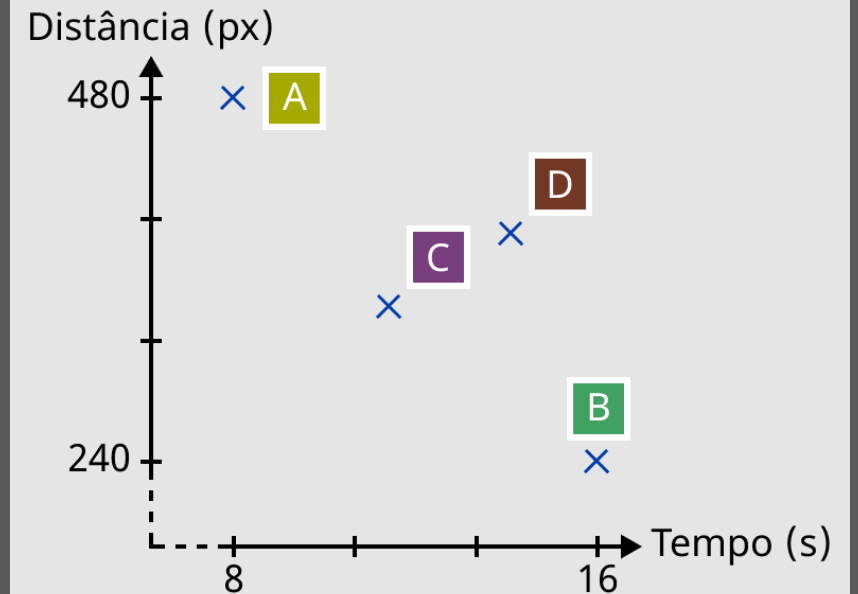
- Quais dos caminhos é o melhor?
- Se tivermos mais caminhos para comparar?

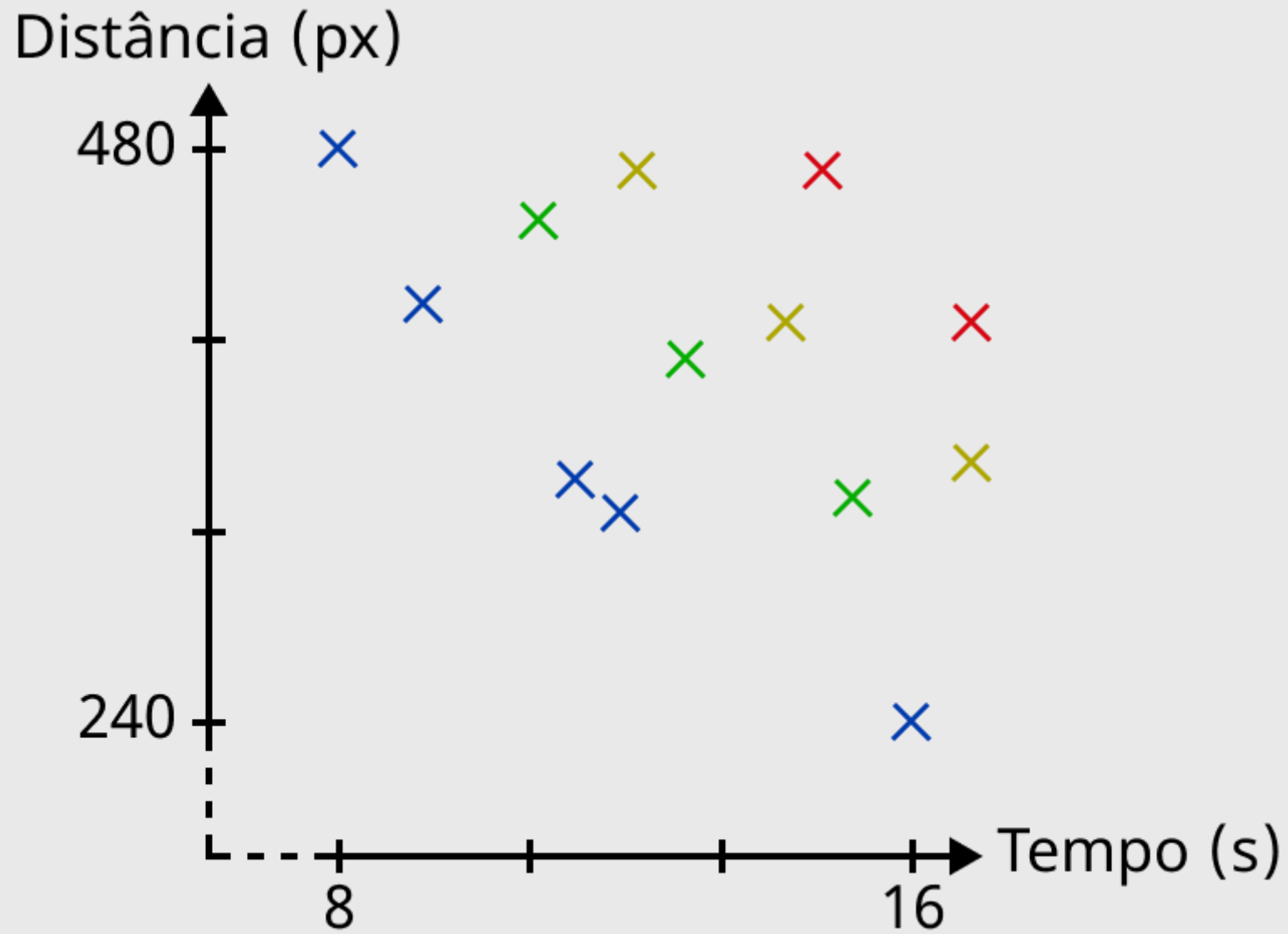


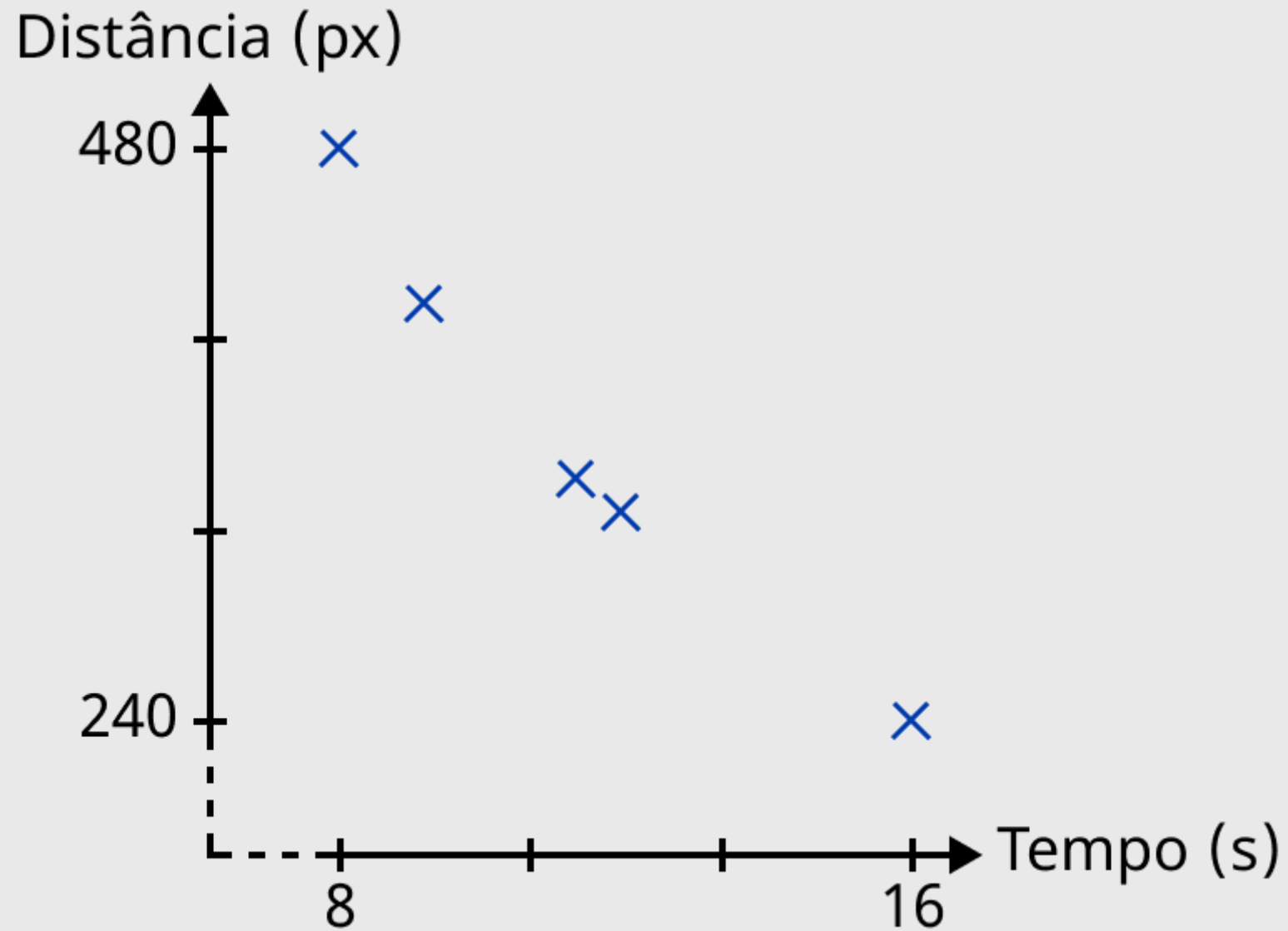
Como combinar as métricas?

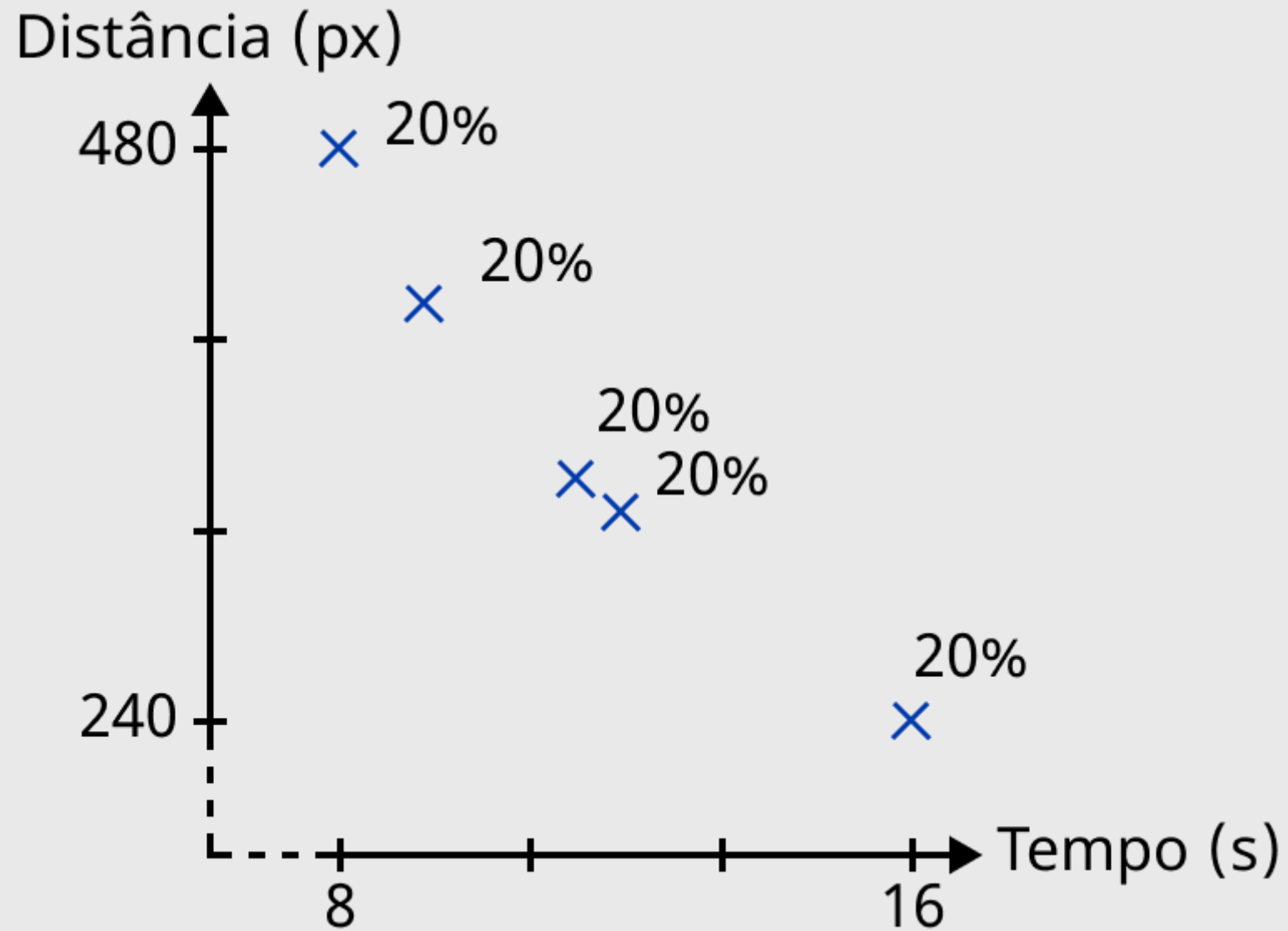


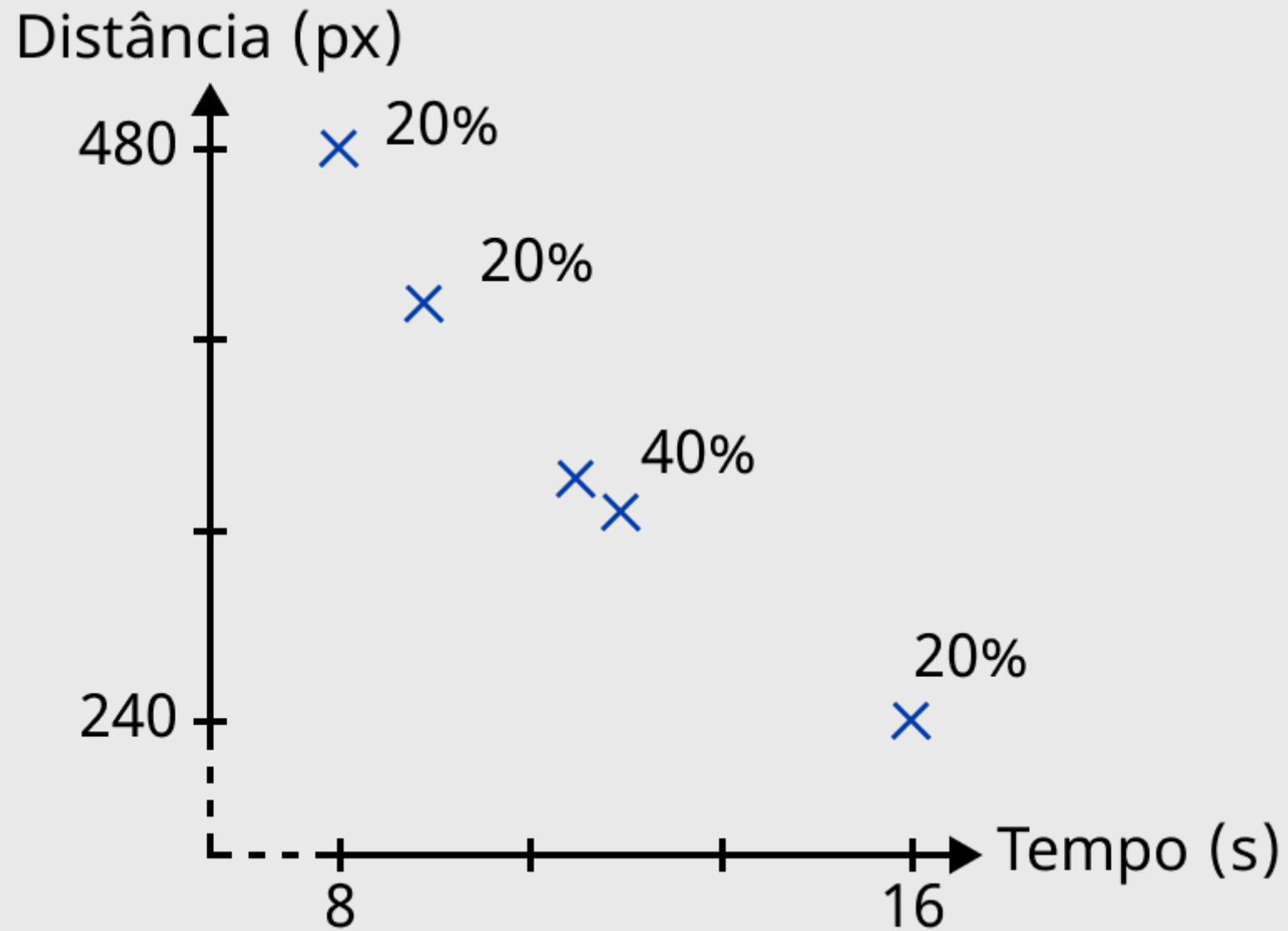
- Quais dos caminhos é o melhor?
- Se tivermos mais caminhos para comparar?

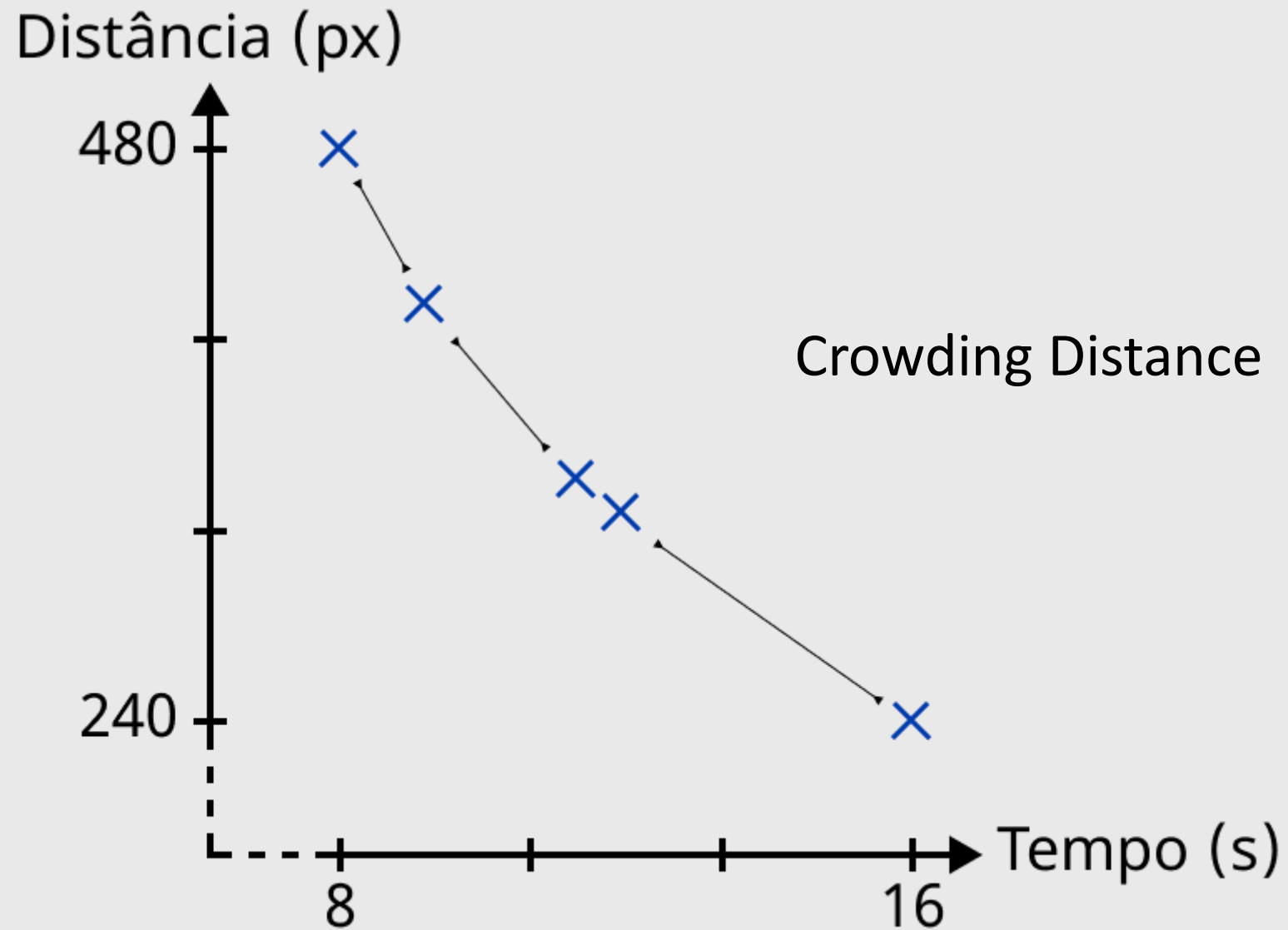












Avaliação Multi-Objetivo

- Ordenar os elementos com maior rank de forma crescente;
- Posteriormente, ordena-los pelo *Crowding Distance* de forma decrescente.

Qual a solução final?



Seleção por Torneio



- Todos os indivíduos tem a mesma probabilidade de serem sorteados.
- São sorteados dois indivíduos diferentes para o torneio.
- O indivíduo com o menor rank será escolhido;
- Em caso de empate, o que apresentar a menor distância será escolhido.

Cruzamento

Parent A [1, 3, 5, 4, 6, 7, 0, 2, 9, 8]

Parent B [1, 5, 6, 7, 2, 4, 8, 9, 3, 0]

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

Parent A	[1, 3, 5, 4, 6, 7, 0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4, 8, 9, 3, 0]

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]
Offspring A	[1, 3, 5, 4, 6, 7,	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]
Offspring A	[1, 3, 5, 4, 6, 7, 8, 9, 3, 0]	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]
Offspring A	[1, 3, 5, 4, 6, 7, 8, 9, 3, 0]	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]
Offspring A	[1, 3, 5, 4, 6, 7,	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1 , 5 , 6 , 7 , 2, 4 ,	8, 9, 3 , 0]
Offspring A	[1, 3, 5, 4, 6, 7,	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1, 3, 5, 4, 6, 7,	0, 2, 9, 8]
Parent B	[1 , 5 , 6 , 7 , 2, 4 ,	8, 9, 3 , 0]
Offspring A	[1, 3, 5, 4, 6, 7, 2, 8, 9, 0]	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

	Head	Tail
Parent A	[1 , 3, 5 , 4 , 6 , 7 ,	0, 2 , 9, 8]
Parent B	[1, 5, 6, 7, 2, 4,	8, 9, 3, 0]
Offspring A	[1, 3, 5, 4, 6, 7,	2, 8, 9, 0]
Offspring B	[1, 5, 6, 7, 2, 4,	

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

Cruzamento

Parent A [1, 3, 5, 4, 6, 7, 0, 2, 9, 8]

Parent B [1, 5, 6, 7, 2, 4, 8, 9, 3, 0]

Offspring A [1, 3, 5, 4, 6, 7, 2, 8, 9, 0]

Offspring B [1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

- O método tem objetivo criar novos indivíduos a partir do material genético dos pais;
- A técnica utilizada depende da representação cromossômica usada.

```
public static Individual DoCrossover(Individual individualA, Individual individualB, int crossoverPosition)
{
    crossoverPosition = crossoverPosition == -1
        ? 1 + random.nextInt(individualA.sequence.size() - 2)
        : crossoverPosition;

    List<Integer> offspringSequence = new ArrayList<Integer>();
    for (int i = 0; i < crossoverPosition; i++){
        offspringSequence.add(individualA.sequence.get(i));
    }

    List<Integer> appeared = new ArrayList<Integer>();
    for (int i = 0; i < offspringSequence.size(); i++){
        appeared.add(offspringSequence.get(i));
    }

    for (int town : individualB.sequence){
        if (appeared.contains(town)){
            continue;
        }

        offspringSequence.add(town);
    }

    return new Individual(offspringSequence);
}
```

Mutação

Troca

[1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

Rotação

[1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

Mutação

Troca

[1, ⁵, 6, 7, 2, ⁴, 3, 0, 9, 8]

Rotação

[1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

Mutação

Troca

4
[1, , 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 5, 6, 7, 2, 4, 3, 0, 9, 8]

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, , 3, 0, 9, 8]
5, 6, 7, 2, 4

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 4, , 3, 0, 9, 8]
5, 6, 7, 2,

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 4, 2, , 3, 0, 9, 8]
5, 6, 7,

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 4, 2, 7, , 3, 0, 9, 8]
5, 6,

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 4, 2, 7, 6, , 3, 0, 9, 8]
5,

Mutação

Troca

[1, 4, 6, 7, 2, 5, 3, 0, 9, 8]

Rotação

[1, 4, 2, 7, 6, 5, 3, 0, 9, 8]

```
private static Individual doSwapMutate(Individual individual){  
    List<Integer> sequence = individual.sequence;  
  
    int[] towns = GetUniqueTowns(individual.sequence);  
  
    Utility.SwapInPlace(sequence, towns);  
  
    return new Individual(sequence);  
}
```

```
public static void SwapInPlace(List<Integer> arr, int[] indexs){  
    int temp = arr.get(indexs[0]);  
    arr.set(indexs[0], arr.get(indexs[1]));  
    arr.set(indexs[1], temp);  
}
```

```
public static Individual DoRotateMutate(Individual individual)
{
    int[] towns = GetUniqueTowns(individual.sequence);

    int firstIndex = towns[0] < towns[1] ? towns[0] : towns[1];
    int secondIndex = towns[0] < towns[1] ? towns[1] : towns[0];

    ArrayList<Integer> newSequence = new ArrayList<Integer>();
    for (int i = 0; i < firstIndex; i++){
        newSequence.add(individual.sequence.get(i));
    }

    ArrayList<Integer> middle = new ArrayList<Integer>();
    for (int i = secondIndex; i >= firstIndex; i--){
        middle.add(individual.sequence.get(i));
    }

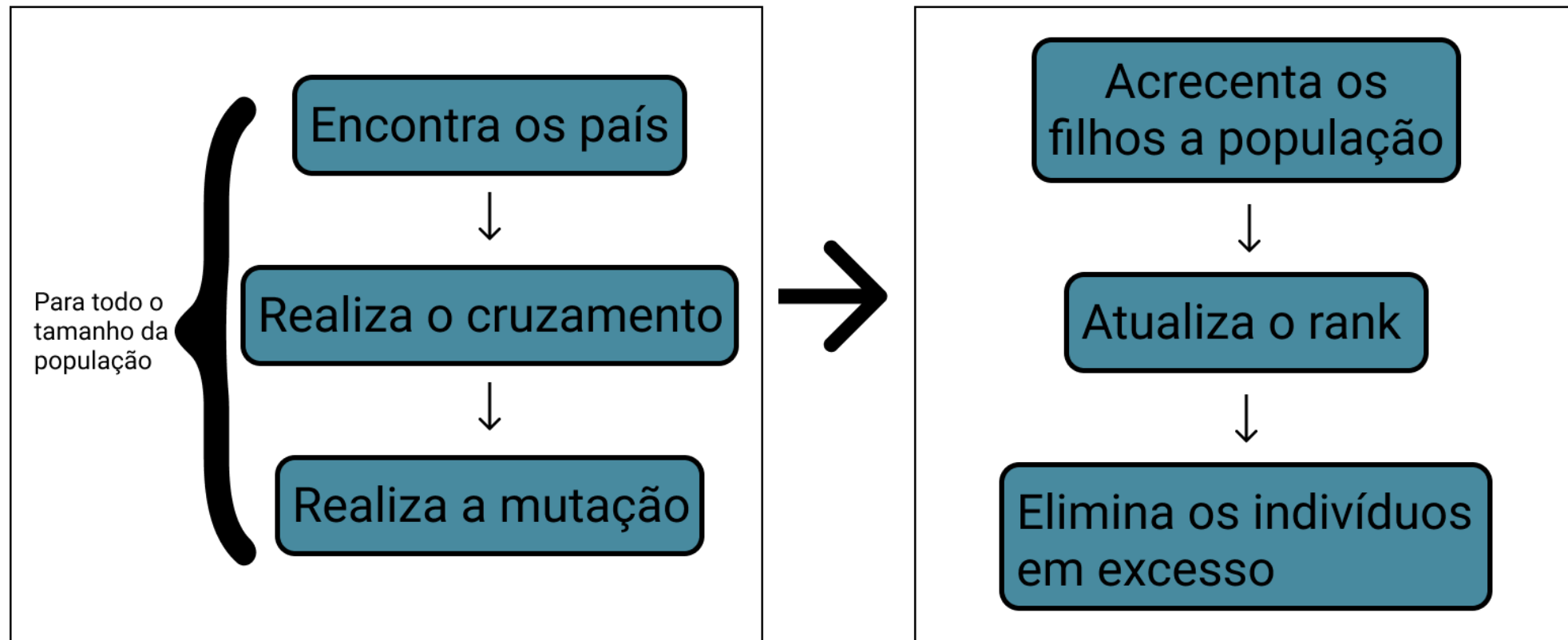
    ArrayList<Integer> tail = new ArrayList<Integer>();
    for (int i = secondIndex + 1; i < individual.sequence.size(); i++){
        tail.add(individual.sequence.get(i));
    }

    newSequence.addAll(middle);
    newSequence.addAll(tail);

    return new Individual(newSequence);
}
```

Estrutura geral do Algoritmo Genético

Em cada geração





Tutorials with Gary



thematheusls