# O Problema do Caixeiro-Viajante - Parte 4

Aluno: João Vitor Venceslau Coelho

## Executando a Metaheurística

Executando a metaheurística GA (*Genetic Algorithm*) implementada na instância ***ulysses22*** utilizando os seguintes parâmetros:

- Qtd. Indivíduos no Torneio para Crossover: **10**
- Taxa de Cruzamento (Cross Rate): **0.8**
- Taxa de Mutação (Mutation Rate): **0.1**
- Tamanho da População: **1000**
- Quantidade de épocas: **3000**
- Tempo máximo de execução: **2 horas**
- Melhor solução inicial: **gerada por algoritmo guloso**

Além do seguinte parâmetro para reprodutibilidade:

- Seed de geração de números pseudo-aleatórios: **42**

E os seguintes critérios de parada:

- Número de épocas alcançou o máximo estabelecido;
- Tempo de execução do algoritmo alcançou o máximo estabelecido;

Foi obtido o seguinte resultado:

- Tour: **[11, 9, 10, 19, 20, 21, 1, 8, 18, 4, 22, 17, 2, 3, 16, 13, 14, 12, 7, 6, 15, 5]**
- Com custo: **7087**
- Após executar por **52** minutos e **36** segundos
- Após **3000** iterações

## Descrição do Algoritmo Implementado

O algoritmo *Genético* implementado é uma adaptação do algoritmo apresentado em aula para o problema do Caixeiro Viajante com a adoção de algumas das ideias apresentadas durante o curso. Abaixo está o pseudocódigo do algoritmo junto a uma breve explicação.

```
Genetico(contest_size, cross_rate, mutate_rate, size, epochs, max_time, melhor_solucao ):
  se melhor_solucao igual a NULL faça:
    melhor_solucao ← gerar_tour_aleatorio
  solucao ← copiar melhor_solucao
  populacao ← gerar população aleatória de tamanho size-1
  incluir melhor_solucao  na populacao
  enquanto (passo atual menor que o máximo e tempo de execução do algoritmo menor que o máximo) faça:
    para cada indivíduo da populacao, faça:
      parceiro ← selecionar o melhor parceiro entre uma amostra de tamanho contest_size da populacao
      se sortear valor entre 0 e 1 menor que cross_rate, faça:
        filhos ← realizar cruzamentos entre indivíduo e parceiro
      se sortear valor entre 0 e 1 menor que mutate_rate, faça:
        filhos ← realizar mutações nos filhos
      adicionar filhos na nova populacao
      comparar filhos com a melhor_solucao e atualizar melhor_solucao
    ordenar nova população com base no custo de cada individuo
```

```
   populacao ← melhores size individuos da nova populacao
retorne melhor_solucao
Cruzamento(individuo_a, individuo_b, filho):
  inicio ← sortear uma posição do tour do individuo_a
  final ← sortear uma posição do tour do individuo_a
  se inicio maior que final, faça:
    troque o inicio pelo final e o final pelo inicio
  copia o fragmento do tour do individuo_a definido pelas posições inicio e final para o
começo do tour do filho
  o restante do tour do filho é composto pelos nós do tour do individuo_b que não estão no
fragmento extraído do individuo_a, na ordem que aparecerem
  retornar filho

Mutação(individuo):
  taxa ← 1.0 / tamanho do tour do individuo
  para cada nó do tour do individuo, faça:
    se sorteio de valor entre 0 e 1 menor que taxa, faça:
      sortear um nó do tour do individuo
      trocar as posições do nó atual com o nó sorteado
  retornar individuo
```

As principais diferenças do algoritmo implementado para a versão implementada em aula são:

- Adição do tempo de execução como critério de parada;
- Todos os indivíduos possuem pelo menos 2 descendentes, pois a cada época, à medida em que os indivíduos vão sendo percorridos é selecionada uma amostra da população onde é feito um torneio com o objetivo de selecionar um parceiro para gerar os filhos;
- Como para cada indivíduo as amostras selecionadas da população para o torneio são formadas por elementos aleatórios da população, há uma maior probabilidade dos melhores indivíduos serem selecionados mais vezes como parceiros, pois os torneios irão retornar o melhor indivíduo das amostras selecionadas;
- Após percorrer todos os indivíduos da população atual, é feita a ordenação da nova população em uma lista, como é preferível manter a mesma quantidade inicial de indivíduos, os novos indivíduos (filhos) são ordenados com base na qualidade deles em que os que têm melhor qualidade serão selecionados para compor a população da próxima época;
- Em relação ao tipo de Cruzamento (Crossover), foi utilizada a ideia de selecionar um fragmento de um dos pais, sendo feita a cópia dos elementos que não foram usados ainda na ordem em que aparecem no parceiro, para a formação do filho;
- No tocante ao tipo de mutação, foi feita apenas a troca de posição entre dois elementos aleatórios do filho, onde cada elemento do filho é percorrido e possui probabilidade igual de ser trocado por outro. Essa probabilidade é definida por *1 / tamanho do indivíduo;*

# (EXTRA) Comparação VNS - GRASP - GA

Utilizando os mesmos algoritmos implementados no trabalho anterior, foi feita a comparação da performance deles com o GA. Segue a configuração usada no VNS e no GRASP:
Os parâmetros utilizados no VNS foram:

- Tamanho da vizinhança: **4**
- Iterações máximas: **1500**
- Máximo de Iter. sem melhora: **750**
- Seed: **42**
- Tempo máximo: **2 horas**

Os parâmetros utilizados no GRASP foram:

- *alpha* inicial: **0.0**
- Tamanho da vizinhança: **3**
- Iterações máximas: **7500**
- Máximo de Iter. sem melhora: **2500**
- Iter. antes do incremento do *alpha*: **10**
- Tempo máximo:**2 horas**

Os parâmetros utilizados no GA foram os mesmos apresentados para o ***ulysses22*****,** indicados anteriormente. Tanto o VNS como o GRASP usaram a mesma solução inicial.

Segue a tabela com a execução dos dois algoritmos em 17 das 30 instâncias pesquisadas:

| Nome | Melhor Custo | Greed | Custo VNS | Tempo VNS | Custo GRASP | Tempo GRASP | Custo GA | Tempo GA | Melhor Alg. |
|---|---|---|---|---|---|---|---|---|---|
| burma14 | | 4048 | 3323 | 44s | 3490 | 3m 11s | 3323 | 37m 8s | VNS |
| ulysses16 | 6859 | 9988 | 6909 | 1m 15s | 6972 | 5m 1s | 6859 | 41m 15s | GA |
| gr17 | | 2187 | 2085 | 1m 35s | 2085 | 6m 13s | 2085 | 42m 59s | VNS |
| gr21 | | 3333 | 3303 | 3m 41s | 2986 | 12m 51s | 2707 | 50m 32s | GA |
| ulysses22 | 7013 | 10586 | 7953 | 4m 42s | 7681 | 15m 20s | 7087 | 52m 35s | GA |
| gr24 | 1276 | 1553 | 1332 | 6m 43s | 1430 | 20m 49s | 1323 | 1h 6m 4s | GA |
| fri26 | 937 | 1112 | 1034 | 8m 50s | 937 | 27m 22s | 955 | 1h 10m 12s | GRASP |
| bayg29 | 1610 | 2005 | 1838 | 13m 56s | 1742 | 40m 31s | 1610 | 1h 25m 25s | GA |
| bays29 | 2020 | 2258 | 2111 | 13m 44s | 2037 | 40m 28s | 2036 | 1h 25m 30s | GA |
| dantzig42 | | 956 | 857 | 1h 14m 22s | 836 | 2h | 734 | 2h | GA |
| swiss42 | | 1630 | 1418 | 1h 11m 47s | 1399 | 2h | 1273 | 2h | GA |
| att48 | 10628 | 12861 | 12218 | 1h 57m 14s | 11864 | 2h | 10921 | 2h | GA |
| gr48 | 5046 | 6098 | 5780 | 1h 57m 14s | 5774 | 2h | 5207 | 2h | GA |
| hk48 | | 13181 | 12696 | 1h 55m | 12137 | 2h | 11929 | 2h | GA |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 36s | | | | | |
| eil51 | 426 | 511 | 505 | 2h | 488 | 2h | 437 | 2h | GA |
| berlin52 | 7542 | 8980 | 8279 | 2h | 8156 | 2h | 8275 | 2h | GRASP |
| brazil58 | 25395 | 30774 | 29288 | 2h | 27043 | 2h | 26803 | 2h | GA |

* Os espaços vazios na coluna **Melhor Custo** indicam que não possui um valor conhecido

Os resultados obtidos com o GA são bem melhores que o VNS e o GRASP, porém o tempo gasto é bem maior. O GA foi melhor em 13 das 17 instâncias testadas.