INF 1771
Inteligência Artificial

CAIO VALENTE
GABRIELLE BRANDEMBURG

CVRP
Problema de Roteamento de Veículos Capacitados

29/04/2019

SUMÁRIO

I - Introdução
- I.1 - Algoritmos escolhidos para o CVRP
- I.2 - Algoritmos de geração de solução inicial
- I.3 - Estruturas da solução

II - Desenvolvimento do trabalho
- II.1 - Simulated Annealing
- II.2 - Busca Local Hill Climbing

III - Análise dos resultados (a definir)
- III.1 - Resultados gerais
- III.2 - Comparações gráficas

IV - Conclusão

Hill-climbing
- Relocate (1 elemento) (Inter-rota)
- 2-opt (Intra-rota)
- Best-first

Simulated Annealing
- Relocate (1 elemento)
- Inversão (~2-opt)
- Swap (1-1 elemento)
- Escolha aleatória de vizinhança
- Best-first

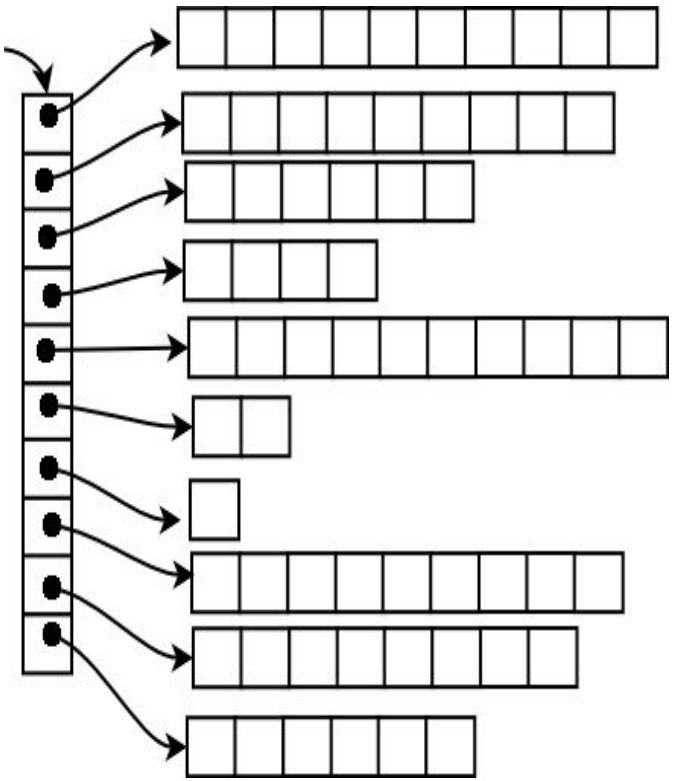
Greedy

- Criação de rotas escolhendo o nó mais próximo do último nó inserido

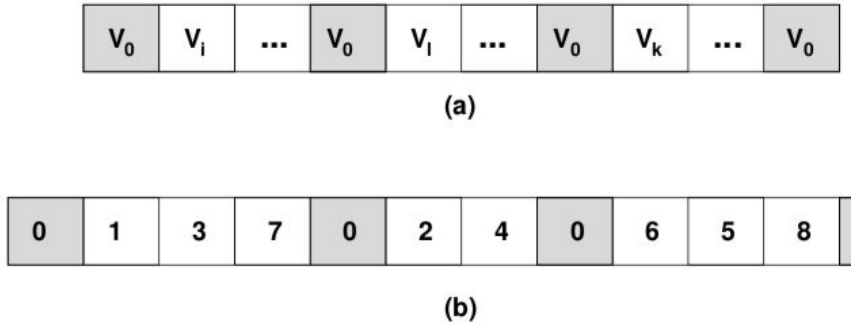
Random

- Criação de rotas escolhendo o nós aleatórios que ainda não estão inseridos em nenhuma rota

Hill-climbing



Simulated Annealing

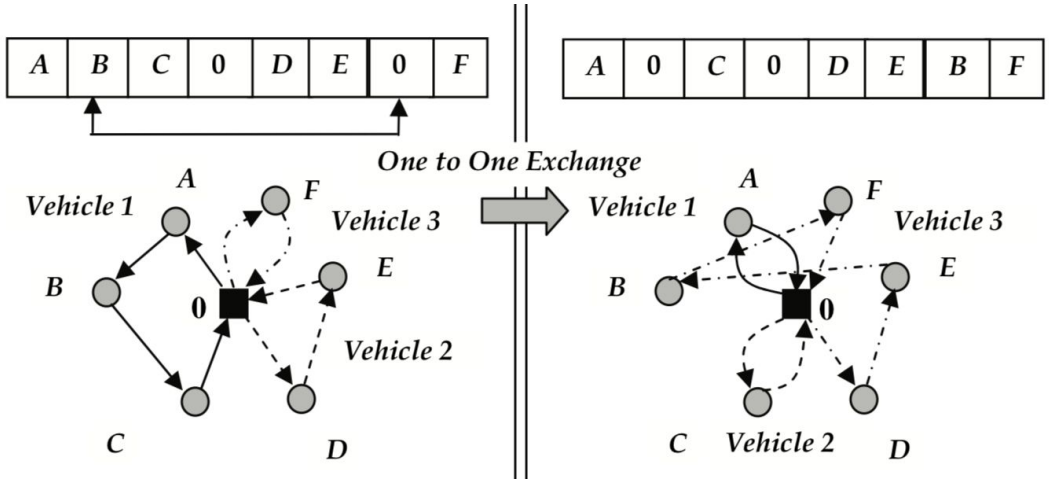


II - Desenvolvimento do trabalho

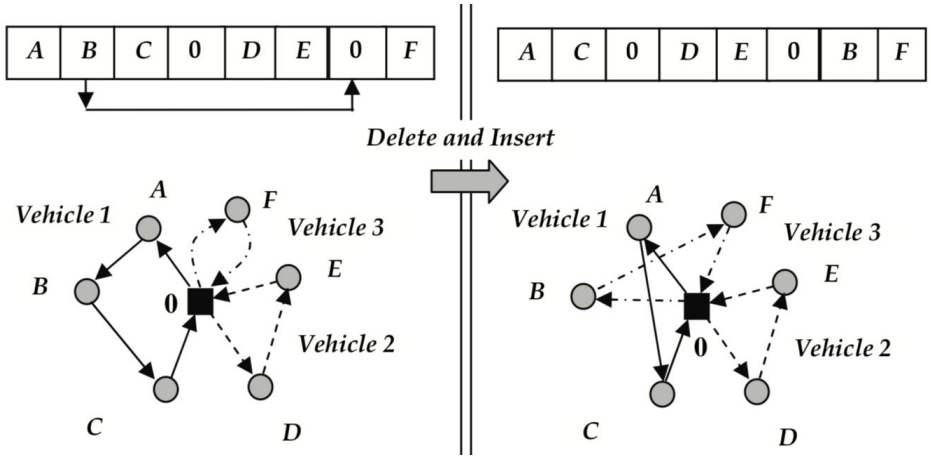
II.1 - Simulated Annealing


```
simulated_annealing():
      solucao_inicial = gera_solucao_inicial()
      solucao_corrente = solucao_opt = solucao_inicial

      ENQUANTO T > TEMP_FINAL:
            ENQUANTO i < VIZINHANCA:
                  novo_vizinho = gera_vizinho(solucao_corrente)

                  SE novo_vizinho não é válido:
                        pula iteração

                  deltaC = custo(novo_vizinho) - custo(solucao_corrente)

                  SE deltaC < 0:
                        solucao_corrente = novo_vizinho
                        SE novo_vizinho é melhor do que solucao_opt:
                              solucao_opt = novo_vizinho
                  SE NÃO:
                        aceita novo_vizinho como solucao_corrente com uma probabilidade exp(deltaC/T)
                  i = i + 1

            T = T*FATOR_T
      RETORNA solucao_opt
```

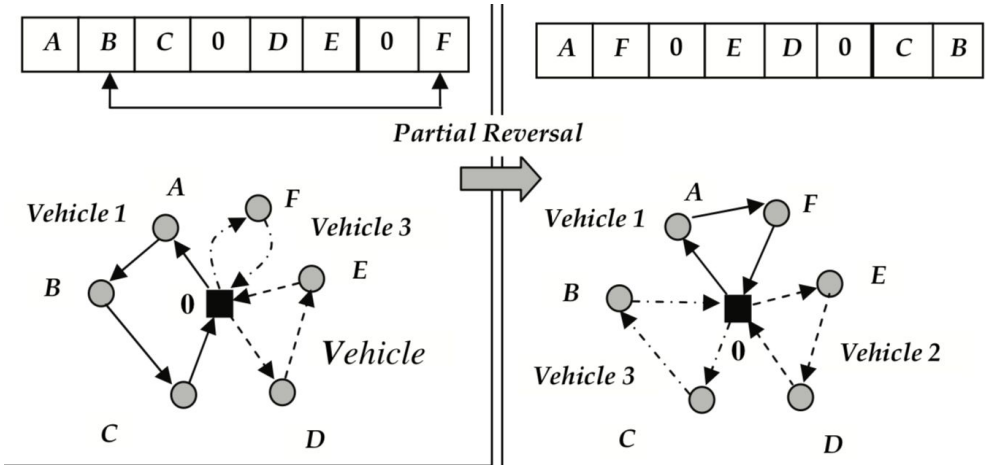
```
gera_vizinho(solucao):
      vizinho = aleatorio(swap(solucao), relocate(solucao), 2_opt(solucao))

swap(solucao_input):
      i,j = duas posições aleatórias de solucao_input
      novo_vizinho = troca nó da posição i com nó da posição j de solucao_input
      RETORNA novo_vizinho

relocate(solucao_input)
      i,j = duas posicoes aleatórias de solucao_input
      novo_vizinho = move nó da posicao i para a posicao j de solucao_input
    RETORNA novo_vizinho

reverse(solucao_input)
      i,j = duas posicoes aleatórias de solucao_input
      novo_vizinho = inverte a sequencia que vai de i até j de solucao input
      RETORNA novo_vizinho
```



SWAP



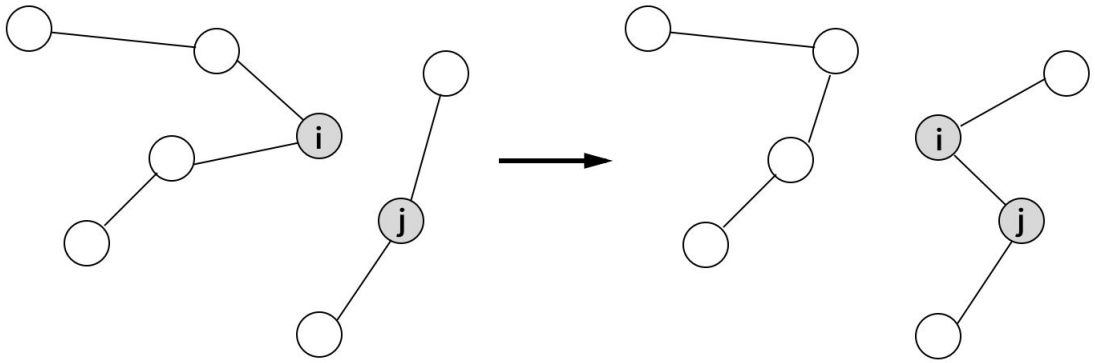
RELOCATE


A B C 0 D E 0 F
A F 0 E D 0 C B
Partial Reversal
Vehicle 1
Vehicle 3
Vehicle
Vehicle 1
Vehicle 2
Vehicle 3

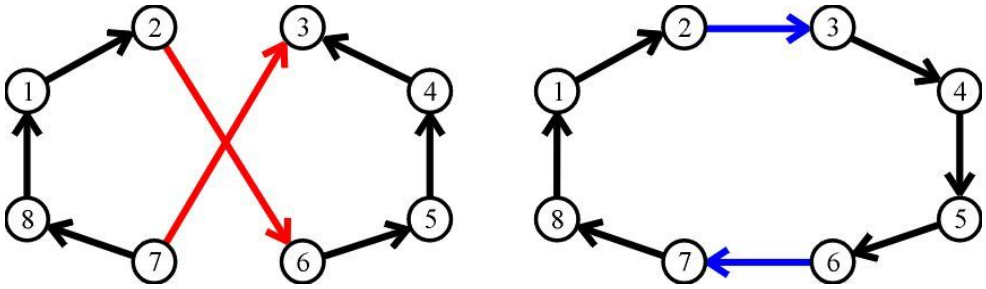
INVERSÃO

```
busca_local():
      solucao_atual = gera_solucao_inicial()
      LOOP:
            vizinho1 = gera_vizinho_inter_rotas(solucao_atual)
            vizinho2 = gera_vizinho_intra_rotas(solucao_atual)

            vizinho = escolhe o melhor vizinho entre vizinho1 e vizinho2

            SE custo(vizinho) é pior ou igual que custo(solucao_atual):
                  PARA
            SE NÃO:
                  solucao_atual = vizinho
      RETORNA solucao_atual
```

```
gera_vizinho_inter_rotas(solucao_atual):
      PARA TODO par (rota1, rota2) na solução_atual:
            PARA TODO par de clientes (i, j):
                  novo_vizinho = remove cliente i da rota1 e coloca antes de cliente j da rota2

                  SE custo(novo_vizinho) é melhor que cust(solucao_atual)
                        RETORNA novo_vizinho
      RETORNA solucao_atual

gera_vizinho_intra_rotas(solucao_atual):
      PARA CADA rota_corrente da solucao_atual:
            PARA TODO par de clientes (i, j) [tal que i < j] da rota_corrente:
                  novo_vizinho = inverte sequência que vai de i até j da rota_corrente

                  SE custo(novo_vizinho) é melhor que custo(solucao_atual)
                        RETORNA novo_vizinho
      RETORNA solucao_atual
```



RELOCATE



2-OPT

III - Análise dos resultados

III.1 - Resultados gerais

X-n101-k25	RESULTADO	ERRO RELATIVO [%] (BKS)	TEMPO (s)
BKS	27591	-	-
HILL-CLIMBING (GREEDY)	40346	31.61	5.723
SA (GREEDY)	32375	14.78	2.715
HILL-CLIMBING (RANDOM)	45744	39.68	429.835
SA (RANDOM)	35179	21.57	2.380

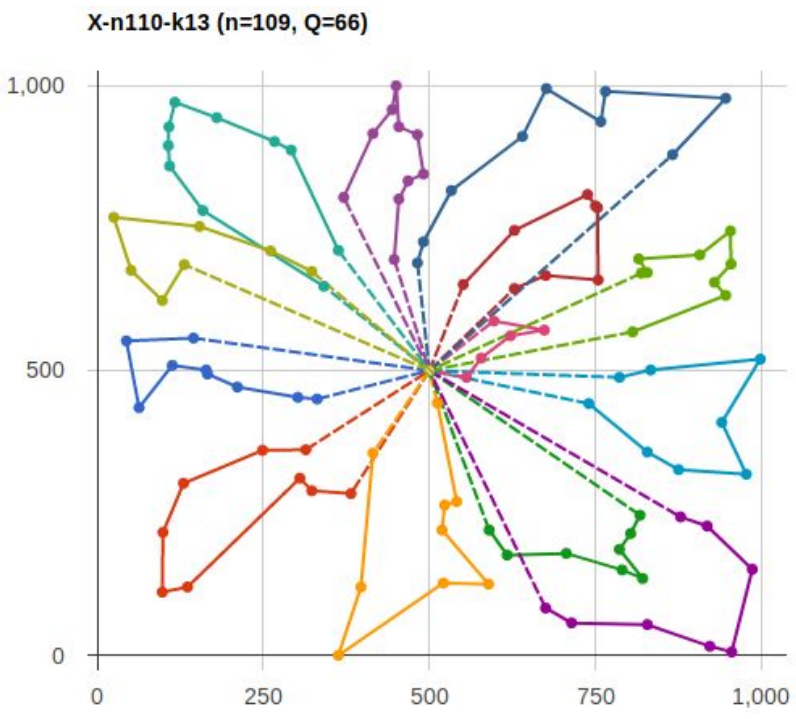
X-n110-k13	RESULTADO	ERRO RELATIVO [%] (BKS)	TEMPO (s)
BKS	14971	-	-
HILL-CLIMBING (GREEDY)	19139	21.78	4.275
SA (GREEDY)	17925	16.48	2.127
HILL-CLIMBING (RANDOM)	34845	57.04	1931.241
SA (RANDOM)	24531	38.97	1.862

III - Análise dos resultados

III.1 - Resultados gerais

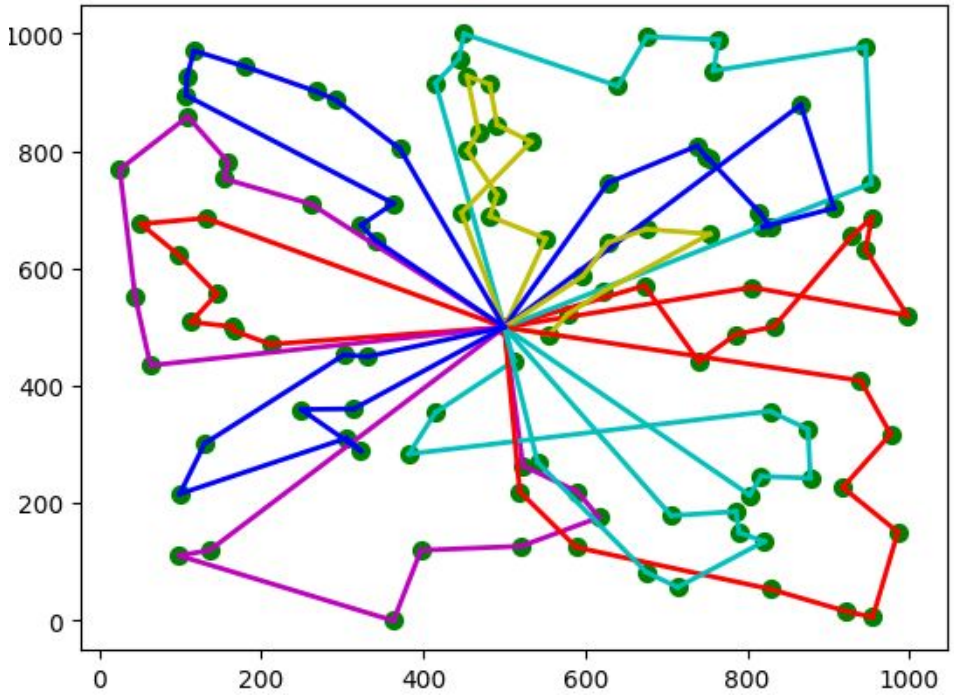
X-n115-k10	RESULTADO	ERRO RELATIVO [%] (BKS)	TEMPO (s)
BKS	12747	-	-
HILL-CLIMBING (GREEDY)	17172	25.77	22.261
SA (GREEDY)	16663	23.50	1.764
HILL-CLIMBING (RANDOM)	27422	53.52	2272.781
SA (RANDOM)	23719	46.26	1.564

X-n204-k19	RESULTADO	ERRO RELATIVO [%] (BKS)	TEMPO (s)
BKS	19565	-	-
HILL-CLIMBING (GREEDY)	23551	16.92	83.844
SA (GREEDY)	23328	16.13	4.896
HILL-CLIMBING (RANDOM)	55532	64.77	927.222
SA (RANDOM)	38295	48.91	5.009



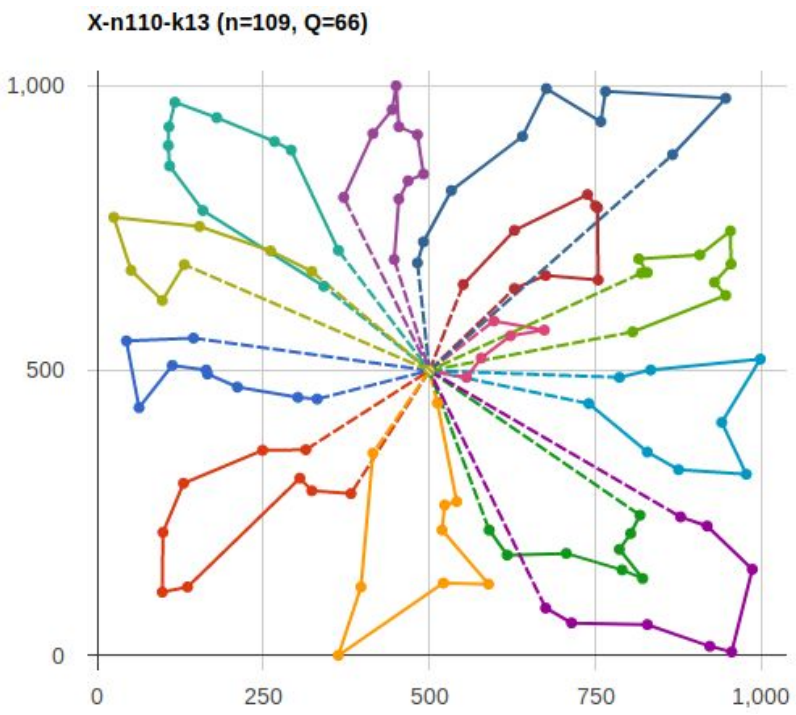
Custo da solução ótima: 14971
Custo da solução inicial: 19244

Annealing, Greedy, 100 times



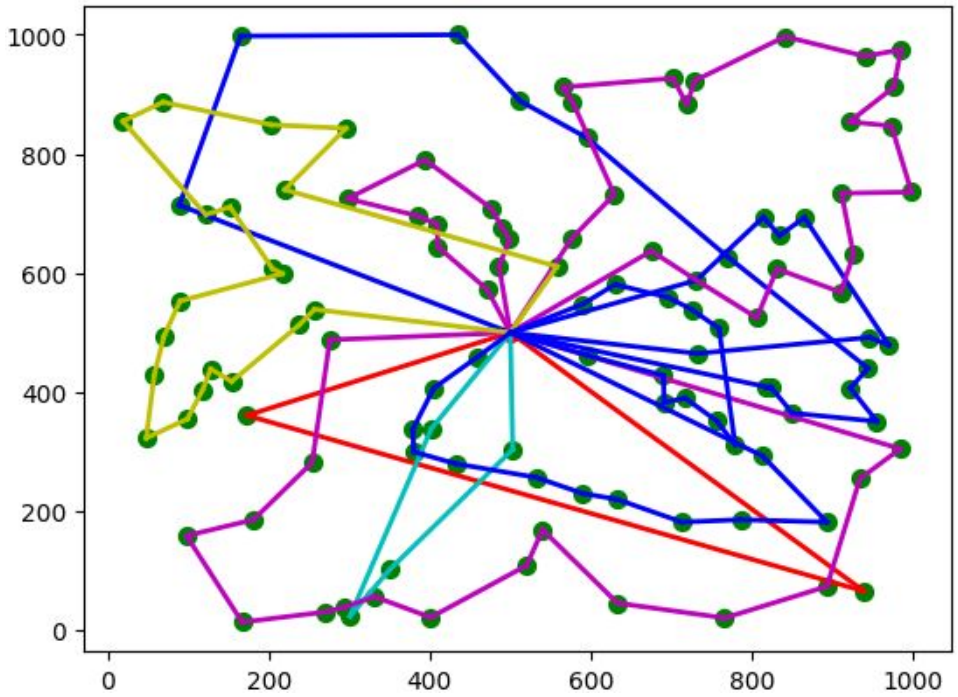
Melhor solução estatística:
Custo: 17118
Tempo: 212.7 segundos

Solução média:
Custo: 17925
Tempo: 2.1 segundos



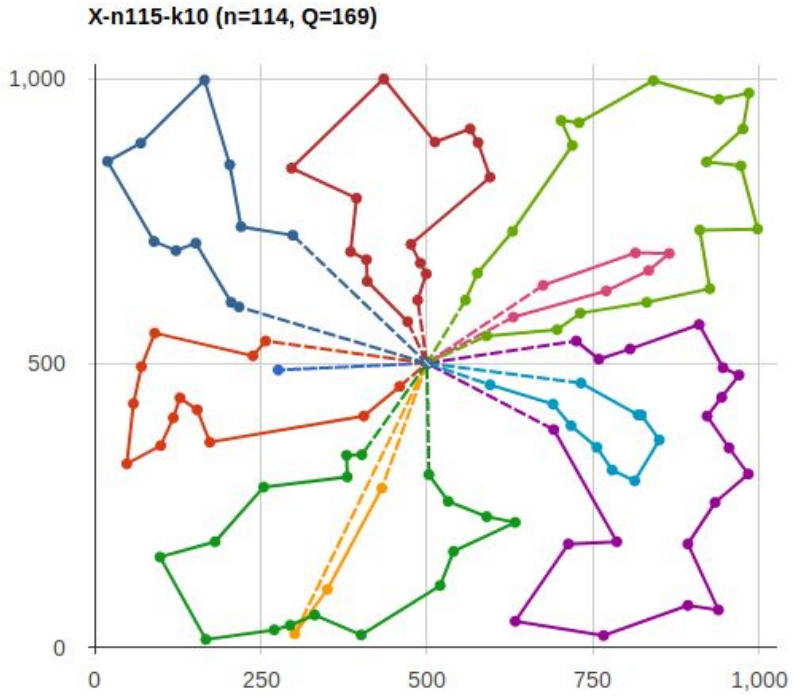
Custo da solução ótima: 14971
Custo da solução inicial: 19244

Annealing, Greedy, 100 times, learning



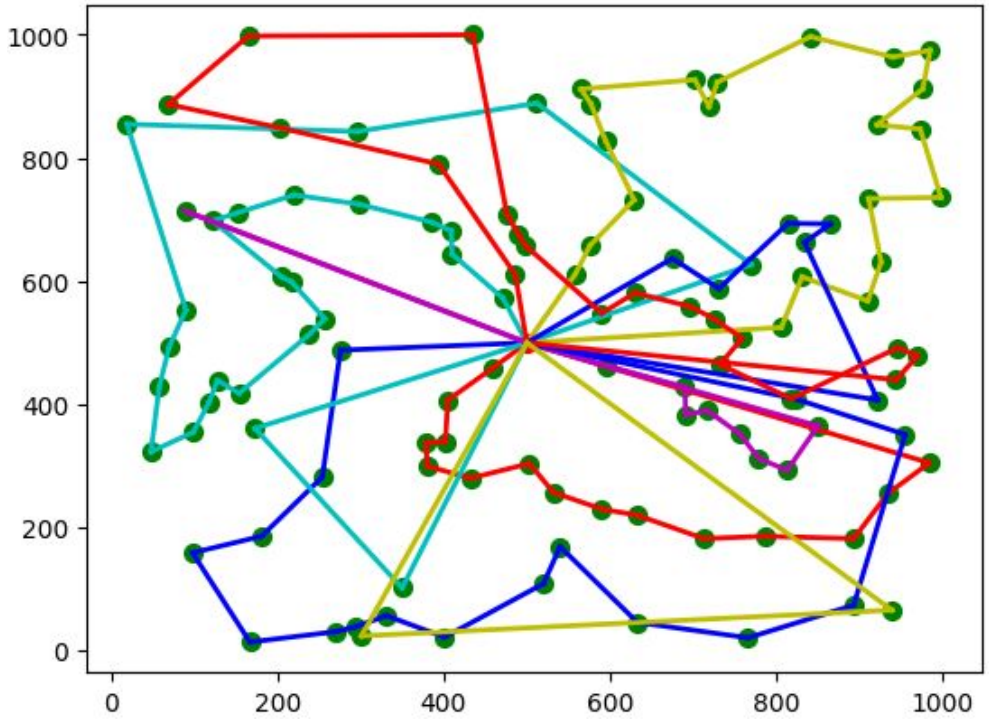
Melhor solução com aprendizado:
Custo: 16092
Tempo: 170.1 segundos

Solução média:
Custo: 16112
Tempo: 1.7 segundos



Custo da solução ótima: 12747
Custo da solução inicial: 17711

Busca Local, Greedy, deterministico



Solução final:
Custo: 17172
Tempo: 22.26 segundos

IV - Conclusão

- Diferença grande em relação a BKS
- Possíveis melhorias:
  - Hill-climbing
    - Retornar melhor vizinho entre N primeiros melhores vizinhos
    - Geração de vizinhos mais discrepantes
  - Simulated Annealing
    - Cooling Schedule mais elaborada (taxa de queda dinâmica)
    - Aumentar a quantidade de iterações na vizinhança
    - Usar a mesma estrutura para a solução usada na HC