Relatório Trabalho I – TSP

Felipe Pessoa - 1411716

Pedro Ferreira - 1320981

**Introdução**

Escolhemos para esse trabalho implementar um algoritmo para o problema do Caixeiro Viajante.

Como experiência, resolvemos trabalhar com duas linguagens diferentes, cada um implementando um algoritmo de busca diferente, para no final podermos comparar os resultados.

**Problema**

A entrada para o problema do Caixeiro Viajante é um conjunto de cidades, podendo já ser fornecida as distâncias entre elas ou a posição das mesmas.

Ele se resume em obter um caminho ótimo (de menor custo) de forma que você comece em uma cidade qualquer, passe por todas as outras sem repetir nenhuma, e volte à cidade inicial.

**Metodologia**

* Simulated Annealing (c#) – Felipe
  + O método implementado realiza múltiplas iterações para poder atingir um resultado melhor. Ele também se utiliza de algumas constantes definidas por meio de testes para otimizar a solução encontrada. Todos esses valores estão descritos junto com as soluções nos resultados.
  + O método consiste em iterar várias vezes criando, a cada iteração, uma solução que pode levar a um resultado melhor.
  + A iteração é controlada pela temperatura. A cada iteração, a temperatura é ajustada por um fator de redução.
  + Na primeira iteração, é gerada uma solução aleatória. A partir dessa, uma nova solução é gerada, usando a vizinhança da solução corrente.
  + Foi experimentado primeiro um swap simples, em que é trocada de posição duas cidades na rota. Após ler alguns artigos acadêmicos, ficou claro que o método 2-opt, em que uma parte da rota é invertida, era melhor para esse problema.
  + A cada iteração, o algoritmo deve decidir entre ficar com a solução atual ou trocar para a nova. Para isso, é verificado primeiro se o custo da solução recém gerada é melhor. Caso seja, ela é escolhida. Se não for melhor, ela passa por um método de aprovação, que leva em conta a probabilidade de ela ser escolhida. A conta usa um exponencial com a temperatura, o que faz com que a chance de uma solução pior ser escolhida só diminui conforme a temperatura desce.
  + Para roda o algoritmo com uma seed diferente da usada pela biblioteca, passe como primeiro argumento para o executável.
* Algoritmo Genético ( python) – Pedro.
  + O algoritmo genético implementado utiliza um princípio em que cada geração possui um valor inicial de 30 indivíduos, a partir do momento em que se atinge 70 indivíduos, 10 novos indivíduos aleatórios são adicionados e ocorre a seleção dos 30 melhores para a próxima geração.
  + O termino do algoritmo ocorre quando o número de gerações, previamente informado, for atingido
  + A escolha dos pais é feita de maneira aleatória para que ocorra uma maior variedade de indivíduos na população.
  + A partir do momento em que um crossover acontece, os dois filhos gerados são adicionados à população e podem ser escolhidos como pais dessa mesma geração.
  + O crossover se baseia em encontrar os ciclos presentes nos cromossomos e a mutação, que se torna cada vez menos recorrente com o passar das gerações, se baseia do swap de 2 cromossomos.
  + É necessário a instalação do "numpy" e do "PyQt5".
  + Não foi possível a seleção de uma seed para a aleatoriedade, caso fosse colocada uma seed o valor aleatório seria u=o mesmo para cada chamada de função, o que quebraria o meu algoritmo.

**Resultados**

Os resultados gerados pelos programas estão nos arquivos .sol junto com esse relatório. O nome segue o padrão: nomeDaInstanciaPy.sol e nomeDaInstanciaC#.sol.

Abaixo estão os resultados com os valores usados para gera-los:

* Algoritmo Genético (python)
  + Gr17:
    - Melhor resultado: 2085
    - Caminho encontrado: 2 10 9 1 4 8 11 15 0 3 12 6 7 5 16 13 14
    - Media (10): 2197
    - Tempo: 0.226715087890625
    - Gerações: 100
  + Gr21:
    - Melhor resultado: 2707
    - Caminho encontrado: 9 16 18 19 10 3 11 0 6 7 5 15 4 8 2 1 20 14 13 12 17
    - Media (10): 3103
    - Tempo: 0.7172746658325195
    - Gerações: 200
  + Gr24:
    - Melhor resultado: 1368
    - Caminho encontrado: 0 11 3 22 8 12 13 19 1 14 9 4 7 6 5 23 20 16 18 21 17 2 10 15
    - Media (10): 1479
    - Tempo: 0.9067695140838623
    - Gerações: 300
  + Hk48:
    - Melhor resultado: 13952
    - Caminho encontrado: 43 26 36 18 42 20 32 4 2 9 46 7 5 33 35 39 8 29 22 24 12 27 31 28 25 47 14 1 23 34 16 40 21 17 6 37 13 11 30 19 3 45 44 38 41 10 15 0
    - Media (10): 15644
    - Tempo: 16.692864418029785
    - Gerações: 2000
  + Si175:
    - Resultado: 26653
    - Caminho encontrado: 25 26 27 29 28 3 1 0 145 139 120 132 131 130 129 128 166 155 115 156 114 135 136 137 138 141 140 134 142 143 144 172 173 53 54 56 62 21 74 73 72 91 110 149 148 119 133 121 117 87 59 58 60 65 67 69 68 94 108 126 127 83 82 43 42 89 90 20 19 17 18 55 57 106 92 93 107 10 9 8 122 123 124 125 77 159 170 169 112 147 146 84 47 46 97 80 37 78 12 11 13 14 85 99 98 111 171 167 95 45 44 40 39 70 71 105 104 36 34 32 33 35 38 41 81 96 79 113 158 168 161 157 100 101 86 51 52 64 23 24 22 2 7 6 5 4 153 152 151 88 103 102 30 118 31 66 63 15 16 50 49 48 61 75 76 109 164 116 165 160 154 163 162 174 150
    - Tempo: 397.997136592865
    - Gerações: 10000
* Simulated Annealing (c#)
  + Gr17:
    - Caminho encontrado: 7 5 16 13 14 2 10 9 1 4 8 11 15 0 3 12 6
    - Custo (distância): 2085
    - Fator de redução de temperatura: 0,9983
    - Temperatura inicial: 500
    - Número de iterações: 17
    - Seed: Valor Default do método (valor de semente padrão dependente de tempo)
  + Gr21:
    - Caminho encontrado: 6 0 11 3 10 19 18 16 9 17 12 13 14 20 1 2 8 4 15 5 7
    - Custo (distância): 2707
    - Fator de redução de temperatura: 0,9983
    - Temperatura inicial: 500
    - Número de iterações: 21
    - Seed: Valor Default do método (valor de semente padrão dependente de tempo)
  + Gr24:
    - Caminho encontrado: 11 3 22 8 12 13 19 1 14 18 17 21 16 9 4 20 7 23 5 6 2 10 15 0
    - Custo (distância): 1272
    - Fator de redução de temperatura: 0,9983
    - Temperatura inicial: 500
    - Número de iterações: 24
    - Seed: Valor Default do método (valor de semente padrão dependente de tempo)
  + Hk48:
    - Caminho encontrado: 46 6 37 13 17 11 16 34 30 19 10 15 41 3 45 44 38 43 26 36 18 0 1 14 42 20 32 4 28 25 47 23 40 21 12 27 31 24 2 22 29 8 9 39 35 33 5 7
    - Custo (distância): 11675
    - Fator de redução de temperatura: 0,9983
    - Temperatura inicial: 500
    - Número de iterações: 48
    - Seed: Valor Default do método (valor de semente padrão dependente de tempo)
  + Si175:
    - Caminho encontrado: 124 125 130 129 136 141 138 139 142 97 96 95 76 134 135 133 132 92 131 93 90 94 77 78 140 137 79 1 33 42 35 14 8 46 47 45 44 43 54 31 29 30 13 17 6 5 62 4 37 39 38 32 40 34 36 2 63 3 7 16 61 15 28 27 18 60 19 59 20 58 9 49 50 48 55 41 53 51 70 80 66 67 23 22 21 10 64 25 24 0 12 11 52 57 26 81 82 68 65 86 71 85 83 108 56 143 103 106 105 104 98 99 100 101 102 69 113 145 144 146 147 150 149 110 151 118 160 159 158 116 117 119 89 88 87 120 121 123 164 162 161 157 156 122 166 165 167 170 169 168 163 174 173 172 155 112 154 148 171 152 153 115 114 111 109 107 84 72 73 74 91 75 128 127 126
    - Custo (distância): 25008
    - Fator de redução de temperatura: 0,9983
    - Temperatura inicial: 500
    - Número de iterações: 175
    - Seed: Valor Default do método (valor de semente padrão dependente de tempo)

**Conclusão**

Através da análise dos resultados chegamos à conclusão de que o Simulated Annealing alcançou melhores resultados, tanto em questão de tempo quanto menor distância. O fator randômico fortemente presente no algoritmo genético gerou uma discrepância maior entre resultados de uma mesma instancia de teste, o simulated annealing se mostrou extremamente mais consistente em seus resultados. Em cima de estudos, acreditamos que a substituição da mutação, que é um swap entre duas posições aleatórias, por uma busca local pode fazer com que o algoritmo genético supere os valores de menor distancia encontrada pelo Simulated Annealing, entretanto o tempo de execução continuará sendo pior.