Relatório Trabalho I – TSP

Felipe Pessoa

Pedro Ferreira

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

Como experiência, resolvemos trabalhar com duas linguagens diferentes cada um, cada um implementando uma busca diferente, para no final podermos comparar os resultados. Essas foram:

* Simulated Annealing (c#) – Felipe
  + O método implementado realiza múltiplas iterações para poder atingir um resultado melhor. Ele também utiliza-se de algumas constantes definidas por meio de testes para otimizar a solução encontrada. Todos esses valores estão descritos junto com as soluções nos arquivos com os resultados.
* 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 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.

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

* Resultados (python)
  + Gr17: Melhor resultado: 2085; Media (10): 2197; Tempo: 0.226715087890625; Gerações: 100
  + Gr21: Melhor resultado: 2707; Media (10): 3103; Tempo: 0.7172746658325195; Gerações: 200
  + Gr24: Melhor resultado: 1368; Media (10): 1479; Tempo: 0.9067695140838623; Gerações: 300
  + Hk48: Melhor resultado: 13952; Media(20): 15644; Tempo: 16.692864418029785; Gerações: 2000
  + Si175 – Resultado: 26653; Tempo: 397.997136592865; Gerações: 10000