Avaliação 03
Descritivo de Implementação

O presente documento tem por objetivo servir de guia para a elaboração dos trabalhos da TERCEIRA unidade da componente curricular GRAFOS. Ao fim desta unidade, é esperado que o aluno tenha conhecimento sobre a teoria relativa ao conteúdo abordado, e consiga replicá-la em um ambiente computacional.

Como ficou definido na aula inicial, as atividades avaliativas irão constar de implementações computacionais dos pressupostos teóricos vistos em sala de aula. Tais implementações deverão seguir algumas regras básicas.

(a) As implementações deverão ser feitas em grupo de até 5 participantes (discentes matriculados na turma);

(b) A linguagem a ser adotada é de livre escolha. Contudo, as implementações devem abranger a teoria vista integralmente. O uso de bibliotecas auxiliares é permitido, porém, não para as atividades fim dos algoritmos. Em outras palavras, as estruturas de dados e pseudocódigos devem ser implementados em sua integralidade;

(c) Apesar de ser uma atividade em grupo, a avaliação ocorrerá de forma individual, de forma que todos devem ter conhecimento dos códigos e funções implementadas pelo grupo.

(d) Não custa lembrar que a presença física nas aulas também é objeto de avaliação, contribuindo para a construção da nota individual. Então, evitem faltar.

(e) A data para entrega (virtual) é o dia **28 de janeiro de 2025.**

Deverá ser entregue:

i. **O código utilizado, para inspeção:** Os códigos deverão estar comentados, **em português**, detalhando o funcionamento de cada função/procedimento, incluindo o formato da entrada e qual a saída esperada. (Poderá ser disponibilizado em GitHub ou similar)

ii. **Um vídeo curto**, explicando o funcionamento/estrutura geral do programa, IDE utilizada e os resultados decorrentes da aplicação da função aos dados da avaliação.

iii. **Uma lista de atividades desenvolvidas por integrante**, detalhando a participação efetiva em cada implementação, seja em sua concepção, implementação, revisão ou teste.

Deverá ser implementado:

A. **Para o PROBLEMA DO CAIXEIRO VIAJANTE**

(1) Algoritmo Guloso

(2) Algoritmo da Inserção mais barata

(3) Algoritmo GRASP + Busca Local (pelo menos 2 diferentes)

Os dados e problemas estão descritos no TCC Abdiel. São 12 problemas sendo 6 por distância e 6 por tempo. Está anexado também uma planilha com os dados.

DEVE ser entregue um arquivo em PDF comparando os resultados encontrados com os demais trabalhos existentes (Tabela 5 do TCC). A coluna GLPK informa a solução exata. Nos resultados, informe a rota encontrada, o tempo e o valor da solução. Lembre-se de descrever seu ambiente computacional.

Obs. As técnicas de busca local também podem ser utilizadas como pós-processamento do algoritmo guloso e da inserção mais barata. Contudo, como são algoritmos determinísticos, basta usar a busca local uma única vez. A escolha de quais buscas locais implementar é livre. ***Deve-se descrever no arquivo PDF como foi pensada a busca local****. Existem algumas bem eficientes, como a de Lin-Kernighan (https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic) .*

Algumas sugestões de busca local:

1. Troca de Vizinhos (Swap) – Simples

- **Descrição:** Troca a posição de duas cidades na solução atual.
- **Como funciona:**
  1. Selecione duas cidades **i** e **j** (aleatórias ou sistemáticas).
  2. Inverta suas posições no percurso.
  3. Avalie o custo total do percurso modificado.
  4. Substitua a solução atual se a nova solução for melhor.
- **Exemplo:** Percurso atual: A→B→C→D. Troque B e D: A→D→C→B.
- **Complexidade:** $O(n^2)$ para avaliar todas as combinações possíveis.

---

2. Reversão de Subcaminho (2-opt) – Fácil e Eficiente

- **Descrição:** Remove duas arestas não adjacentes do percurso e reconecta os subcaminhos resultantes de forma invertida para reduzir a distância total.
- **Como funciona:**
  1. Escolha dois índices i e j no percurso.
  2. Inverta a ordem das cidades entre i e j.
  3. Avalie o novo custo do percurso.
  4. Aceite a modificação se ela reduzir o custo total.
- **Exemplo:** Percurso atual: A→B→C→D→E. Inverta entre B e D: A→D→C→B→E.
- **Complexidade:** $O(n^2)$ por iteração completa, mas é muito eficaz para encontrar melhorias.

3. Inserção de Cidades (City Insertion) – Moderada

- **Descrição:** Remove uma cidade do percurso e a insere em uma posição diferente para tentar reduzir o custo total.
- **Como funciona:**
    1. Escolha uma cidade c_i no percurso.
    2. Remova-a temporariamente do percurso.
    3. Tente inseri-la em todas as outras posições possíveis.
    4. Substitua a solução atual se a nova solução for melhor.
- **Exemplo:** Percurso atual: A→B→C→D. Remova C e insira entre D e A: A→B→D→C.
- **Complexidade:** $O(n^2)$ para avaliar todas as possibilidades.

---

4. 3-opt – Avançada

- **Descrição:** Remove três arestas e reconecta os subcaminhos resultantes de todas as formas possíveis, escolhendo a que minimiza o custo.
- **Como funciona:**
    1. Escolha três arestas (a,b),(c,d),(e,f).
    2. Teste todas as maneiras de reconectar os subcaminhos entre os nós a,b,c,d,e,f.
    3. Substitua a solução atual pela nova solução se for melhor.
- **Complexidade:** $O(n^3)$ por iteração, mas é mais poderosa para eliminar laços complexos no percurso.

---

5. Vizinhança de Shuffle (Or-opt) – Complexa

- **Descrição:** Move um bloco de cidades (subcaminho de tamanho k) para outra posição no percurso.
- **Como funciona:**
    1. Escolha um subcaminho $c_i, c_{i+1}, \ldots, c_{i+k}$.
    2. Remova o subcaminho e insira-o em uma posição diferente no percurso.
    3. Avalie o novo custo do percurso.
    4. Substitua a solução atual se a nova solução for melhor.
- **Exemplo:** Percurso atual: A→B→C→D→E. Mova B→C para depois de D: A→D→B→C→E.
- **Complexidade:** Depende do tamanho k; geralmente $O(n^2)$.