Manual de utilização - task 5

Grupo constituinte

- Guilherme Gaspar n.º 2020218933
- Maria Carolina Fernandes n.º 2021218374
- Pedro Nuno Monteiro n.º 2021218544

Manual

Na pasta *Task 5* é possível encontrar 5 ficheiros *.py* (functions.py, menus.py, draw.py, task.py e calculos.py) e uma pasta *networks* com 4 ficheiros de texto *.txt* que correspondem às 4 redes, obtidas da SNDLib.

Ficheiro ***task.py***

- Este é o ficheiro principal, aquele que deve ser corrido na linha de comandos para iniciar o programa através de ***python task.py***
- Antes de executar o programa, é necessário criar uma pasta com o nome "output", de forma a evitar erros durante a execução. Nesta pasta serão guardadas todas as imagens geradas pelo programa, incluindo a imagem da rede inicial, a imagem da rede final com os caminhos definidos e, no caso do algoritmo de Suurballe, todas as imagens correspondentes aos diferentes passos do processo.
- Ao correr o programa, é apresentado um menu com 3 opções
    1. Determinar Caminhos
    2. Cálculos Estatísticos
    3. Sair
- Ao escolher a opção n.º 1, o programa irá apresentar uma lista com as 4 *networks* definidas, podendo o utilizador escolher da 1 à 4 ou selecionar a opção 5, para “Inserir outra rede”. Depois da seleção, aparece um terceiro menu com a lista de nós disponíveis da rede selecionada, criando, ainda, um grafo para ajudar na visualização (que é automaticamente fechado depois da escolha dos nós de origem e destino)
    - Após esta escolha e dependendo do algoritmo selecionado, o programa é direcionado para a função correspondente
        - *find_best_paths* caso Two Step Approach
        - *suurballe* caso Suurballe
        - usar ambos os métodos, onde correm as 2 funções

    - Escolhendo a opção 5, o programa irá apenas pedir ao utilizador para escrever, incluindo a extensão *.txt*, o nome da rede: para isso basta acrescentar na pasta onde estão presentes as restantes redes com a extensão .txt - em seguida, basta escrever o nome do ficheiro adicionado, com a respetiva extensão.
    - No caso de escolha do método de suurballe, este apresenta, para cada passo, o gráfico correspondente; tal opção pode ser “desligada” escolhendo a opção 0 no menu precedente ao

algoritmo, que passará todos os passos à frente e apresentando apenas o grafo final.

- Ao escolher a opção 2, aparece, na mesma, o menu com as várias redes disponíveis, à semelhança do menu anterior.
    - Depois de escolher a rede, aparece um terceiro menu com as opções de calcular
        1. Taxa de resolução
        2. Taxa de resolução ótima
        3. Erro médio do custo
    - Após escolher o que calcular, são apresentados os resultados pedidos.
- Ao escolher a opção 3, o programa é terminado

Ficheiro ***menus.py***

- Este ficheiro foi adicionado ao projeto para melhorar a organização do mesmo. Aqui, estão as funções que apresentam ao utilizador os diferentes menus existentes

1. *show_ask_network()*
    a. Apresenta ao utilizador o menu com as várias redes disponíveis, abrindo o ficheiro correspondente

2. *ask_network()*
    a. Solicita ao utilizador o nome do ficheiro da rede adicional, devolvendo este mesmo nome

3. *ask_origin_destiny()*
    a. Pede ao utilizador os nós de origem e destino, apresentando, para isso, uma lista com todos os nós da rede, devolvendo estes mesmos valores

4. *ask_which_algorithm()*
    a. Apresenta ao utilizador o menu para escolher o algoritmo a utilizar

5. *ask_skip_forward()*
    a. menu que pergunta ao user se pretende passar todos os passos do algoritmo de suurballe à frente

Ficheiro ***functions.py***

- Este ficheiro contém agora 6 funções que servem de suporte ao programa principal.

1. *retrieve_data(data)*
    a. É a função que, ao receber a informação *data* (que corresponde ao conteúdo de um ficheiro *.txt*), vai buscar a informação relativa aos **nós** e aos **arcos** de cada rede.
    b. A função faz esta busca e adiciona diretamente no grafo G linha-a-linha, inicialmente para os nós e depois para os arcos

c. Retorna o grafo **G** e a a tabela ***node_mapping*** serve como tabela de conversão, ou seja, atribui a cada nó, um número para que seja mais fácil para o utilizador escolher o nó de origem e nó de destino.

2. *find_best_paths(G, origem, destino)*
   a. Recebe como parâmetros o grafo G, o nó de origem e o nó destino
   b. Calcula o **caminho mais curto** entre os nós de origem e destino, através do método Djisktra
   c. Em seguida, copiamos o grafo original e removemos todos os nós que pertenciam ao primeiro caminho, e calculamos o **segundo** caminho mais curto
   d. Este segundo caminho mais curto pode não existir se o nó for de primeira ordem ou se for um nó de

3. *suurballe(G, origem_orig, destino_orig)*
   a. Recebe como parâmetros o **grafo G**, o nó de **origem** e o nó de **destino**.
   b. Calcula o caminho mais curto entre origem e destino no grafo original, utilizando o algoritmo de **Dijkstra**.
   c. Aplica a técnica de **node splitting**, dividindo cada nó do caminho principal em dois (ex: A_in, A_out) para permitir caminhos disjuntos.
   d. Volta a mostrar o caminho principal determinado em *b)* com o node splitting.
   e. Copia o grafo e transforma a rede:
      i. Calcula os **custos reduzidos** para todos os arcos com base nas distâncias anteriores.
      ii. **Remove** os arcos do primeiro caminho que vão no sentido inverso à origem.
      iii. **Inverte** os arcos do primeiro caminho (com custo 0).
   f. Calcula um **segundo** caminho mais curto no grafo transformado:
      i. Este caminho pode não existir se a estrutura da rede não permitir dois caminhos disjuntos
   g. **Remove** sobreposições entre os dois caminhos, fazendo o desentrelaçar dos *paths*
   h. **Reconstrói** os caminhos finais e converte os nós divididos de volta para os nomes originais.
   i. Retorna os dois caminhos disjuntos entre origem e destino.

4. *split_nodes(G, source_orig, target_orig)*
   a. Recebe como parâmetros o **grafo G**, o nó de **origem** e o nó de **destino originais**.
   b. Cria um novo grafo direcionado H, onde cada nó do caminho principal é dividido em dois: **nó de entrada** (A_in) e **nó de saída** (A_out).
   c. Adiciona uma aresta com **custo nulo** (cost = 0) entre A_in e A_out para cada nó original.
   d. Define o novo nó de origem (s) como **source_out** e o novo destino (t) como **target_in**.
   e. Retorna o novo grafo H com os nós divididos, a nova **origem s** e o novo **destino t**

5. *merge_split_path(split_path)*
    a. Recebe como parâmetro uma lista de nós **split_path** correspondente a um caminho no grafo com node splitting
    b. Cria uma nova lista chamada **original_path**, onde vai armazenar o caminho convertido para os nomes dos nós originais.
    c. Para cada nó no caminho dividido:
        i. Remove o sufixo **_in** ou **_out**, recuperando o nome original do nó.
        ii. Adiciona esse nome à lista original_path, garantindo que não haja repetições consecutivas.
    d. **Retorna** o caminho convertido com os nomes dos nós originais

6. *is_valid_path(path, original_path)*
    a. Função que verifica se um caminho no grafo com node splitting corresponde a um caminho válido
    b. É apenas utilizado para verificar o 2.º caminho determinado no algoritmo de suurballe.
    c. Retorna True se for válido e Falso caso contrário.

Ficheiro ***draw.py***

- Este ficheiro embarca as diferentes funções utilizadas para os desenhos dos grafos - é uma junção de algumas funções previamente definidas noutros ficheiros, com a adição da nova função para desenho do suurballe
- Em relação à versão 3.0, acata apenas alguns novos parâmetros de entrada, que permitem algum controlo no desenho de cada grafo e nas mensagens que aparecem no terminal
- As alterações aparecem a sublinhado

1. *draw_empty_network(G, node_mapping)*
    a. Recebe como argumentos o grafo G e a tabela de conversão node_mapping
    b. Antes de ser pedido os nós origem e destino, é apresentado o grafo G, com os respectivos nós já numerados, de forma a que os utilizadores possam antes de escolher, observar a rede na sua totalidade.

2. *draw_network(G, node_mapping, origem, destino, caminho_tsa, caminho2, caminho_sur, caminho3, algoritmo)*
    a. Desenha o grafo destacado até três caminhos e os nós de origem e de destino
    b. Guarda o grafo como imagem em "output/Rede Final.png"

3. *draw_suurballe(G, origem_split, destino_split, caminho1_split, caminho2_split, filename)*
    a. Recebe como parâmetros: o **grafo G**, a **origem_split,** o **destino_split,** lista de nós do **primeiro** e **segundo caminho** e o nome do **arquivo** para guardar o gráfico.

b. Para facilitar essa visualização, utilizamos a função draw_suurballe, que gera gráficos interativos e guarda as imagens representando cada passo do algoritmo.
    i. Mostra o estado do grafo após cada etapa de execução do Suurballe
    ii. Destaca os caminhos encontrados durante o cálculo, permitindo perceber o que foi mudado
    iii. Ilustra a transformação do grafo, incluindo a divisão dos nós no formato _in/_out
    iv. Guarda o ficheiro na localização "output/nome_etapa"

Ficheiro ***calculos.py***

- Este ficheiro contém 4 funções que permitem realizar os cálculos necessários para a análise da tarefa 4.

1. *calculos_auxiliares(G, otimo, calcular_erro_medio)*
    a. função que realiza todos os cálculos necessários à tarefa 4, servindo de suporte às restantes funções
    b. Esta função retorna vários valores que são importantes para os calculos das funções seguintes:
        i. ***pares*** - número de pares origem destino da nossa rede. Assumimos que os pares são não direcionados, logo o número de pares será sempre (n x n-1).
        ii. ***pares_validos*** - número de pares onde onde ambos TSA e Suurballe funcionaram. Esta variável serve só para termos uma ideia de quantas vezes é que ambos os métodos funcionam pois são estes casos que contam para o cálculo do erro médio.
        iii. ***resolvidos_tsa*** - número total de pares resolvidos pelo TSA, ou seja, número de vezes que para um par origem destino o TSA encontrou solução.
        iv. ***resolvidos_sur*** - número total de pares resolvidos pelo Surballe, ou seja, número de vezes que para um par origem destino o Surballe encontrou solução.
        v. ***resolvidos_otimos*** - número total de soloções ótimas encontradas pelo TSA
        vi. ***erro_medio*** - valor do erro médio de custos

2. *calculo_taxa_resolusao(G)*
    a. através do return da função anterior, apresenta um menu ao utilizador com os resultados relativos à taxa de resolução do TSA e do suurballe.

3. *calculo_taxa_resolusao_otima(G)*
    a. através do return da função *calculos_auxiliares()*, apresenta um menu ao utilizador com os resultados relativos à taxa de resolução ótima.

4. *calculo_erro(G)*
    a. através do return da função *calculos_auxiliares()*, apresenta um menu ao utilizador com os resultados relativos ao erro médio do TSA.