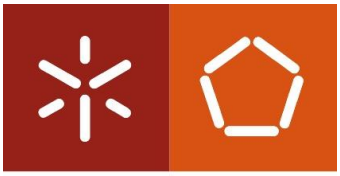

Universidade do Minho

Departamento de Informática

Inteligência Artificial (3º ano de LEI)

Trabalho Prático

2ª Fase

Relatório de Desenvolvimento

Grupo 13

Ariana Lousada (a87998) Miguel Gomes (a93294)

Rita Lino (a93196) Rui Armada (a90468)

4 de janeiro de 2022

Resumo

O presente trabalho prático foi desenvolvido no âmbito da unidade curricular Inteligência Artificial. O principal objetivo deste projeto consistiu em estimular o uso de técnicas de formulação de problemas e na aplicação de diversas estratégias para a sua resolução com a utilização de algoritmos de procura. Ao longo deste documento vai ser explicado todo o processo de desenvolvimento desde o início do projeto até à solução final concebida pela equipa de trabalho.

Capítulo 1

Introdução

O projeto desenvolvido nesta segunda fase consiste na implementação de um sistema de recomendação de circuitos de entrega de encomendas através da utilização do trabalho desenvolvido na primeira fase.

Este sistema de circuitos irá ter como base um grafo constituído pelo nodo inicial(ponto de recolha dos estafetas, i.e. o centro de distribuição da *Green Distribution.*), pontos de entrega e nodo final(que coincide com o nodo inicial).

Com isto, foram utilizados vários algoritmos de procura de grafos, a partir dos quais foram realizados testes de *performance*. Todas as decisões e raciocínio aplicados pela equipa de trabalho serão expostos ao longo deste documento.

Capítulo 2

Preliminares

De modo a ser possível fazer pesquisas no grafo a construir, foi necessário escolher vários algoritmos de pesquisa, construindo predicados para os adaptar em Prolog.
Algoritmos de procura não informada:

- Pesquisa em Profundidade;
- Busca Iterativa Limitada em Profundidade;

Algoritmos de procura informada:

- A estrela;
- Gulosa;

A *performance* destes algoritmos irá ser testada com base no tempo, espaço e custo da sua execução, assim como a qualidade da solução encontrada.

Capítulo 3

Manual de utilização do sistema de reconhecimento

Para ser possível executar o sistema desenvolvido é necessário em primeiro lugar instalar o Python. A instalação pode ser feita através da consulta do seguinte link : `https://www.python.org/downloads/`.

Para instalar as dependências do programa desenvolvido, apenas é necessário executar o seguinte ficheiro:

```
./python.sh
```

Este comando é necessário para instalar todas as bibliotecas utilizadas no projeto.

Capítulo 4

Descrição do Trabalho

Para o desenvolvimento da solução do problema, foi proposta a aplicação de várias funcionalidades, que vão ser expostas nesta secção do documento, assim como o raciocínio aplicado para a construção do grafo.

4.1 Construção do grafo

De modo a ser possível construir um grafo com a informação da base de conhecimento criada na primeira fase do projeto, foi desenvolvido um *script* em *Python*.

Este programa começa por construir as arestas e os nodos do grafo, armazenando-os em ficheiros separados: `arestas.pl` e `nodos.pl`, respetivamente. Ambos os nodos e as arestas são construídas através do conhecimento contido no ficheiro das encomendas(`encomendas.pl`). As arestas são constituídas pela sua origem, destino e custo(distância), enquanto que cada nodo é constituído pelo seu identificador e listas de encomendas endereçadas(entregues e não entregues) à localização representada pelo nodo.

De seguida é feita a junção dos nodos e das arestas, construindo um grafo final e armazenando-o no ficheiro `graph.pl`.

```
1 %grafo([aresta(Src,Dest,Cost)]).
2 grafo([aresta(0,23,1),aresta(1,9,7),aresta(1,13,8),aresta(2,3,1),aresta(2,6,2),
      aresta(2,24,7),
3     aresta(4,12,11),aresta(4,20,3),aresta(4,21,5),aresta(5,15,9),aresta(5,16,8),
      aresta(5,24,6),
4     aresta(6,19,2),aresta(6,23,8),aresta(7,24,15),aresta(8,18,2),aresta(9,17,6),
      aresta(9,19,7),
5     aresta(9,20,4),aresta(11,20,15),aresta(14,18,12),aresta(14,19,5),aresta
      (14,21,12),
6     aresta(16,23,1),aresta(17,19,8),aresta(17,21,13),aresta(18,19,5),aresta
      (20,21,9)]).
```

4.2 Funcionalidades

Para a solução do problema foi proposta a implementação de seis funcionalidades distintas. Estas funcionalidades vão ser expostas nesta secção, com exceção da segunda que engloba a representação dos diversos pontos de entrega em forma de grafo, que já foi abordada na secção anterior(4.1).

4.2.1 Gerar circuitos de entrega

Para esta funcionalidade o objetivo consiste em gerar os circuitos de entrega que cubram um determinado território.

Utilizando o predicado `solveAll`, todos os circuitos existentes do grafo são calculados. O resultado consiste numa concatenação das listas dos caminhos calculados pelos quatro algoritmos de procura.

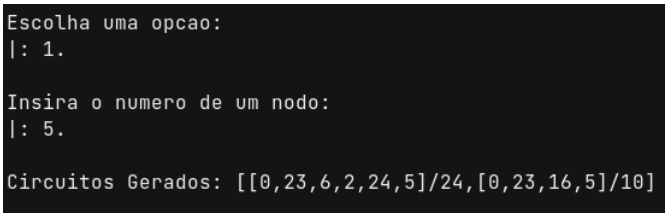

Figura 4.1: Execução da primeira funcionalidade.

4.2.2 Identificação dos circuitos com maior número de entregas

Para esta funcionalidade o objetivo consiste em identificar os circuitos com maior número de entregas por volume e peso.

Em primeiro lugar, percorreram-se todos os circuitos existentes do grafo através do algoritmo de pesquisa em profundidade, criando uma lista de triplos, na qual cada um contém o número de encomendas, total do peso e total de volume das encomendas entregues por circuito.

Para a ordenação dessa lista utilizou-se o algoritmo *QuickSort*, que ordena a lista por ordem decrescente de número de encomendas por circuito.

```
Escolha uma opcao:
|: 3.

Circuitos com maior numero de entregas:

{  Caminho: [0,23,16,5,24,2,6,19,18,14,21,4,20,9,1,13],
         Numero de Encomendas: 52,
         Peso Total: 2267.23,
         Volume Total: 2427.11  };
{  Caminho: [0,23,16,5,24,2,6,19,17,9,20,4,21,14,18,8],
         Numero de Encomendas: 52,
         Peso Total: 2228.76,
         Volume Total: 2277.76  };
{  Caminho: [0,23,16,5,24,2,6,19,14,21,4,20,9,1,13],
         Numero de Encomendas: 51,
         Peso Total: 2196.26,
         Volume Total: 2406.87  };
{  Caminho: [0,23,16,5,24,2,6,19,9,20,4,21,14,18,8],
         Numero de Encomendas: 51,
         Peso Total: 2207.05,
         Volume Total: 2267.47  };
{  Caminho: [0,23,16,5,24,2,6,19,18,14,21,4,20,9,1],
         Numero de Encomendas: 50,
         Peso Total: 2207.05,
         Volume Total: 2267.47  };
{  Caminho: [0,23,16,5,24,2,6,19,18,14,21,17,9,20,4,12]
         Numero de Encomendas: 49,
         Peso Total: 2228.76,
         Volume Total: 2277.76  };
{  Caminho: [0,23,16,5,24,2,6,19,17,21,4,20,9,1,13],
         Numero de Encomendas: 49,
         Peso Total: 2115.11,
         Volume Total: 2199.99  };
{  Caminho: [0,23,16,5,24,2,6,19,14,21,4,20,9,1],
         Numero de Encomendas: 49,
         Peso Total: 2136.08,
         Volume Total: 2247.23  };
{  Caminho: [0,23,16,5,24,2,6,19,17,9,20,4,21,14,18],
         Numero de Encomendas: 48,
```

Figura 4.2: Execução da terceira funcionalidade.

4.2.3 Comparação de circuitos de entrega

Para esta funcionalidade o objetivo consiste em comparar os circuitos de entrega tendo em conta os indicadores de produtividade.

Em primeiro lugar, construíram-se dois predicados: um que calcula a velocidade perdida por veículo através do peso de encomenda e outro que calcula o menor objeto do segundo elemento de um par.

De seguida calcularam-se todos os caminhos existentes do ponto de origem até ao destino, selecionando o de menor custo, i.e. o de menor distância, tendo também em conta o transporte mais ecológico e o tempo restante para a entrega da encomenda.

Este predicado retorna uma lista com as soluções encontradas por cada um dos quatro algoritmos mencionados na secção 2 deste documento.

```
Escolha uma opcao:
|: 4.

Insira o seu Destino:
|: 23.

Insira o Peso da encomenda:
|: 4.

Insira o Tempo:
|: 3.
Veiculo: bicicleta
Caminho: [0,23]
Distancia: 1
```

Figura 4.3: Execução da quarta funcionalidade.

4.2.4 Escolha do circuito mais rápido

Para esta funcionalidade o objetivo consiste em escolher o circuito mais rápido em termos de distância.

Em primeiro lugar é construída uma lista de caminhos utilizando o predicado desenvolvido para solução da funcionalidade da secção 4.2.1. De seguida é calculado o caminho de menor custo. Uma vez que neste caso o custo corresponde à distância, o circuito de menor custo vai corresponder ao mais rápido.

```
Escolha uma opcao:
|: 5.

Insira o numero do nodo final:
|: 23.

Circuito mais rapido, utilizando o criterio de distancia: [0,23]/1
```

Figura 4.4: Execução da quinta funcionalidade.

4.2.5 Escolha do circuito mais ecológico

Para esta funcionalidade o objetivo consiste em escolher o circuito mais ecológico em termos de tempo.

Em primeiro lugar é construída uma lista de caminhos utilizando o predicado desenvolvido para solução da funcionalidade da secção 4.2.1.

A partir dessa lista, é calculado o caminho cujo custo associado corresponda ao menor. A partir deste custo, é calculada a velocidade final utilizando um valor de peso inserido pelo utilizador.

Por fim, com a velocidade final calcula-se o tempo de entrega, dividindo o custo(que corresponde à distância) pela velocidade, multiplicando o valor resultante por 60, de modo a converter a solução em minutos.

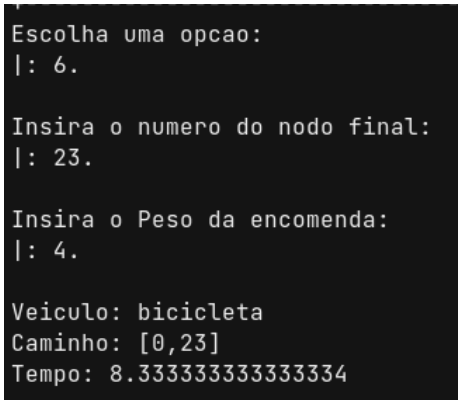

Figura 4.5: Execução da sexta funcionalidade.

4.3 Extras

4.3.1 Menu

De modo a oferecer uma utilização mais amigável do programa ao utilizador, construiu-se um menu de utilização.

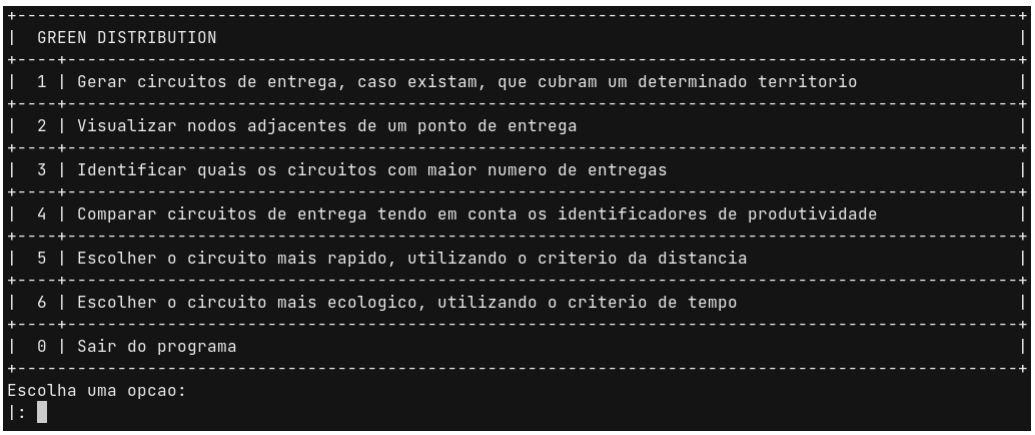

Figura 4.6: Menu de utilização

4.3.2 Funcionalidade extra: Consulta dos pontos de entrega adjacentes

De modo a permitir ao utilizador verificar os pontos de entrega adjacentes a um determinado nodo, apenas é necessário selecionar a opção 2 do menu de utilizador.

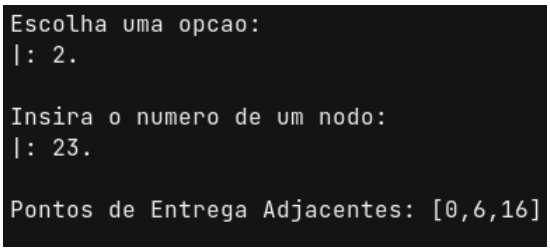

Figura 4.7: Funcionalidade 2 do menu de utilizador.

4.3.3 Visualização de grafos

De modo a ser possível observar os grafos construídos com maior facilidade, desenvolveu-se uma função `visualize` no *script* de `Python` criado que constrói um gráfico utilizando as informações de um dado grafo.

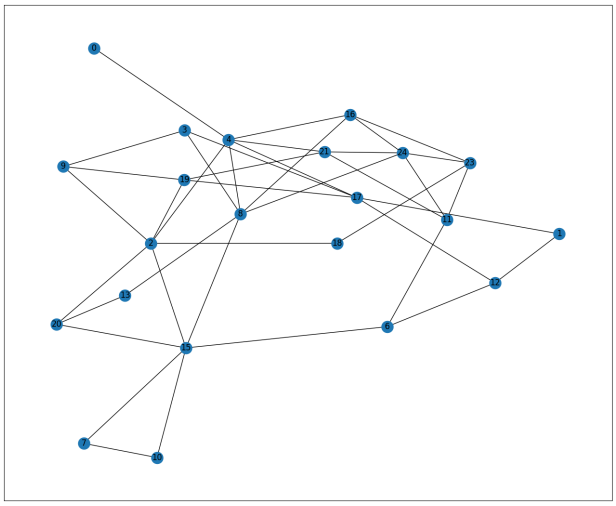

Figura 4.8: Exemplo de um grafo desenhado com a função `visualize`.

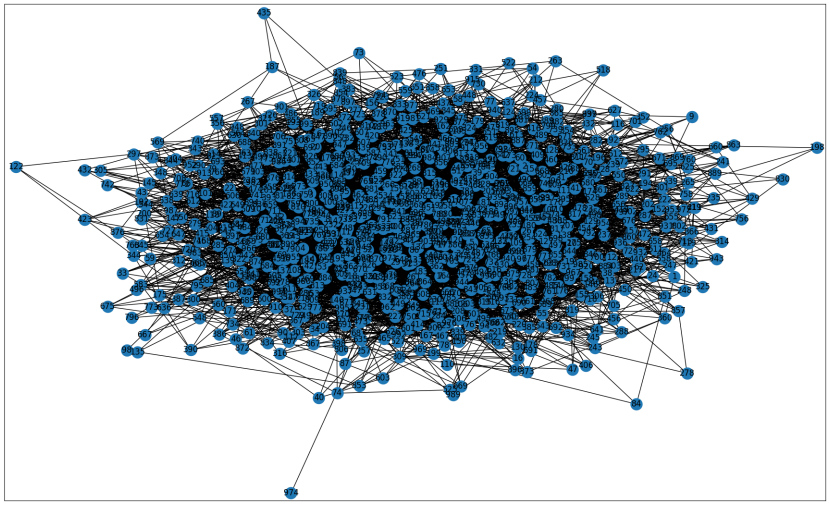

Figura 4.9: Exemplo de um grafo desenhado com a função `visualize`.

Capítulo 5

Resultados

5.1 Testes de *performance*

Para a elaboração de testes de *performance* dos diferentes algoritmos desenvolveu-se o predicado `solveAll` para cada um:

- `solveAllIterativeDeepening`: Para o algoritmo de Pesquisa Iterativa Limitada em Profundidade.
- `solveAllDepthFirst`: Para o algoritmo de Pesquisa em Profundidade (DFS).
- `solveAllAstar`: Para o algoritmo A*.
- `solveAllGreedy`: Para o algoritmo Gulosa.

Cada predicado `solveAll` calcula todos os caminhos desde a origem até um determinado nodo utilizando o algoritmo respetivo. Para todos os testes foram encontrados todos os caminhos desde a origem até ao nodo 23 do seguinte grafo:

```
1 %grafo([aresta(Src,Dest,Cost)]).
2 grafo([aresta(0,23,1),aresta(1,9,7),aresta(1,13,8),aresta(2,3,1),aresta(2,6,2),
      aresta(2,24,7),
3     aresta(4,12,11),aresta(4,20,3),aresta(4,21,5),aresta(5,15,9),aresta(5,16,8),
      aresta(5,24,6),
4     aresta(6,19,2),aresta(6,23,8),aresta(7,24,15),aresta(8,18,2),aresta(9,17,6),
      aresta(9,19,7),
5     aresta(9,20,4),aresta(11,20,15),aresta(14,18,12),aresta(14,19,5),aresta
      (14,21,12),
6     aresta(16,23,1),aresta(17,19,8),aresta(17,21,13),aresta(18,19,5),aresta
      (20,21,9)]).
```

Os testes foram realizados em ambiente *Manjaro(Linux)*, numa máquina com as seguintes características:

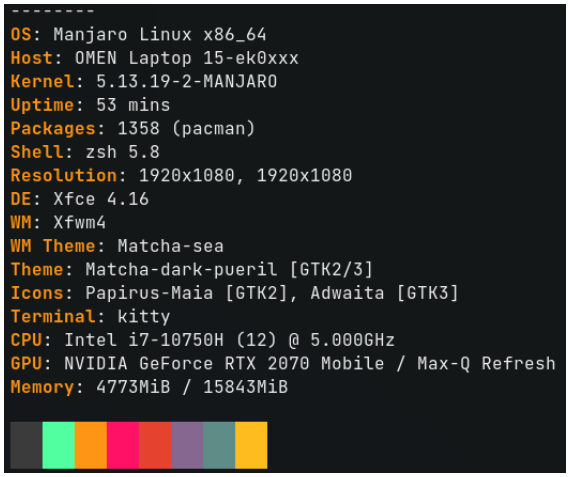

Figura 5.1: Características da máquina utilizada para testes

Variável	Significado
b	fator máximo de ramificação
d	profundidade da melhor solução
m	máxima profundidade da árvore de estados

Estratégia	Tempo(segundos)	Complexidade do Espaço utilizado	Complexidade do Custo (no pior caso)	Encontrou a melhor solução?
Profundidade (DFS)	0.07	$O(b * m)$	$O(b^m)$	Não
Busca Iterativa Limitada em Profundidade	∞	$O(b * d)$	$O(b^m)$	—
Gulosa	0.07	$O(b^m)$	$O(b^m)$	Não
A*	0.08	$O(b^m)$	$O(b^m)$	Sim

5.2 Análise de Resultados

Através dos vários testes realizados, foi possível comparar a execução dos diferentes algoritmos de pesquisa utilizados.

Para o primeiro algoritmo testado, o algoritmo de Pesquisa em Profundidade, foi obtido um tempo de execução de 0.07 segundos. Contudo, este algoritmo é capaz de encontrar a melhor solução possível, uma vez que desenvolve em primeiro lugar os nodos de maior profundidade, sem ter em conta o custo das arestas selecionadas. Uma vez que os grafos utilizados são todos finitos, a equipa de trabalho não se deparou problemas de ciclos infinitos.

Para o segundo algoritmo, o algoritmo de Pesquisa Iterativa em profundidade, não foi possível obter uma solução final, uma vez que o algoritmo entra em ciclo infinito. Uma vez que a solução é sempre a mesma, o `findall` encontra sempre a mesma solução iterativamente, o que impede a conclusão da execução do algoritmo.

Para o terceiro algoritmo, o algoritmo de Pesquisa Gulosa, obteve-se um tempo de execução de 0.07 segundos. Contudo, podem existir casos nos quais não encontra a solução ótima: por exemplo, se existirem informações repetidas no interior do grafo, como nodos e arestas iguais.

Por fim, para o último algoritmo, o algoritmo de Pesquisa A*(estrela), obteve-se um tempo de execução de 0.08 segundos. Este algoritmo encontra sempre a solução ótima do problema sem complicações, apesar de utilizar um grande volume da memória(uma vez que armazena todos os nodos do grafo).

Capítulo 6

Comentários Finais e Conclusão

Após uma análise concisa de todos os testes realizados é possível concluir que o melhor algoritmo para o problema em mãos será o algoritmo de Pesquisa em Profundidade. É de salientar que é possível escolher este algoritmo pelo problema se tratar de um serviço de entregas, que envolve sempre grafos finitos.

Apesar do algoritmo de Pesquisa A*(estrela) ser o mais confiável, não é utilizável em problemas de escala real, uma vez que os serviços de entrega na atualidade envolvem grafos de grande dimensão. Uma vez que este algoritmo armazena **todos** os nodos em memória, é impossível aplicá-lo em sistemas de uma forma eficiente.

Em suma, com a realização deste trabalho prático foi possível avaliar e perceber os desafios de escolha de algoritmos em sistemas cujo o uso de grafos é estritamente necessário.