



Universidade do Minho
Escola de Engenharia – Departamento de Informática

Relatório do Trabalho Prático Individual

Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio

A86788 – Lázaro Pinheiro

Braga, junho 2020

Resumo

Este relatório procura dar suporte ao código desenvolvido para satisfazer as questões colocadas. Para tal, é realizada uma breve contextualização e fundamentação teórica de estratégias de pesquisa em grafos, procurando fundamental e corroborar o trabalho apresentado.

Além disso, é apresentada a abordagem da resolução de cada uma das funcionalidades, recorrendo a imagens ilustrativas dos predicados utilizados.

Por fim, na secção Conclusões e Sugestões, é feita uma pequena reflexão sobre o trabalho realizado, de uma forma geral, assim como as dificuldades sentidas ao longo da sua realização. Procura-se igualmente avançar com algumas sugestões de melhoria.

É importante referir também que na secção Anexos podemos encontrar todos os predicados auxiliares utilizados para a realização das funcionalidades, juntamente com os *outputs* do interpretador.

Tabela de Conteúdos

Resumo	2
Índice de figuras	4
Introdução	5
Preliminares	6
Descrição do Trabalho e Análise de resultados	7
Fundamentação teórica	7
Estratégias de Pesquisa	7
Pesquisa Não-Informada	8
Pesquisa Informada	9
Resolução das funcionalidades propostas	10
Obtenção da Base de Conhecimento	10
Base de Conhecimento.....	11
Desenvolvimento de Requisitos.....	12
Criação de uma API de modo a manipular as pesquisas	14
Execução da Aplicação.....	17
Conclusões e sugestões	18
Referências	18

Índice de figuras

Figura 1 - Tabela de classificação dos Algoritmos de Pesquisa.....	7
Figura 2 - Main do Parser realizado em Java.	10
Figura 3 - Formatação do predicado representativo do facto freguesia.	11
Figura 6 - Formatação do predicado representativo do facto paragem.	11
Figura 7 - Formatação do predicado representativo do facto adjacente.	12
Figura 10 - Excluir um ou mais operadores de transporte para o percurso (DFS).	12
Figura 13 - Escolher o percurso mais rápido (usando critério da distância) (A*).	13
Figura 14 - Escolher o percurso que passe apenas por abrigos com publicidade (DFS).	14
Figura 15 - Escolher o percurso que passe apenas por paragens abrigadas (DFS).	14
Figura 16 - Escolher um ou mais pontos intermédios por onde o percurso deverá passar (DFS).	14
Figura 17 – Definição da interface com o utilizador para o requisito “Calcular um trajeto entre dois pontos”.	14
Figura 18 – Definição da interface com o utilizador para o requisito “Selecionar apenas algumas das operadoras de transporte para um determinado percurso”.	15
Figura 19 – Definição da interface com o utilizador para o requisito “Excluir um ou mais operadores de transporte para o percurso”.	15
Figura 20 – Definição da interface com o utilizador para o requisito “Identificar quais as paragens com o maior número de carreiras num determinado percurso”.	15
Figura 21 - Definição da interface com o utilizador para o requisito “Escolher o menor percurso (usando o critério menor número de paragens)”.	16
Figura 22 - Definição da interface com o utilizador para o requisito “Escolher o percurso mais rápido (usando critério da distância)”.	16
Figura 23 - Definição da interface com o utilizador para o requisito “Escolher o percurso que passe apenas por abrigos com publicidade”.	16
Figura 24 - Definição da interface com o utilizador para o requisito “Escolher o percurso que passe apenas por paragens abrigadas”.	17
Figura 25 - Definição da interface com o utilizador para o requisito “Escolher um ou mais pontos intermédios por onde o percurso deverá passar”.	17
Figura 26 - Exemplo de Interface com o Utilizador.....	17

Introdução

A execução deste trabalho procura um aprofundamento e consolidação dos conteúdos teóricos abordados na Unidade Curricular Sistemas de Representação de Conhecimento e Raciocínio, através de um sistema de pesquisas em grafos. Este trabalho serve-se da linguagem de programação em lógica PROLOG, no âmbito de métodos de Resolução de Problemas e no desenvolvimento de algoritmos de pesquisa, para representar o conhecimento e raciocínio na área do caso de estudo, isto é, um sistema de transportes do concelho de Oeiras.

Desenvolveu-se um sistema, que permite importar os dados relativos às paragens de autocarro, e representa-os numa base de conhecimento, da forma mais adequada. Posteriormente, foi desenvolvido um sistema de recomendação de transporte público para o caso de estudo.

Preliminares

Para a realização deste trabalho prático é necessário conhecer a linguagem de programação PROLOG, competência obtidas no decorrer das aulas de Sistemas de Representação de Conhecimento e Raciocínio, bem como a consolidação de conhecimentos teóricos. É igualmente imperioso possuir conhecimentos sobre estratégias de pesquisa.

De modo a perceber o problema proposto pelo enunciado do trabalho, fez-se uma análise detalhada e minuciosa do mesmo, a fim de obter uma correta interpretação do objetivo pretendido e com a finalidade de obter o conhecimento necessário para se conseguir enveredar pela melhor estratégia de pesquisa consoante a operação a realizar. Foi ainda consultada a bibliografia disponibilizada pelos docentes de modo a aprimorar o conhecimento da temática proposta.

Foi implementado código em Java, com intuito de importar os dados relativos às paragens de autocarro, através do ficheiro Excel fornecido (lista_adjacencias_paragens.xls) para uma base de conhecimento.

Após obtenção da base de conhecimento normalizada e funcional, foram implementados os requisitos espelhados no enunciado do trabalho prático, utilizando a linguagem de programação em lógica PROLOG.

Os fundamentos teóricos estudados para realizar o presente trabalho prático foram:

- Estratégias de Pesquisa:
 - Pesquisa Não-Informada(cega):
 - Pesquisa em Profundidade (DFS);
 - Pesquisa Informada (heurística):
 - A*;

Descrição do Trabalho e Análise de resultados

Numa primeira instância, optou-se por realizar uma pequena fundamentação teórica das temáticas abordadas no tópico anterior, para que melhor se possa compreender o que foi definido na elaboração do trabalho prático.

Fundamentação teórica

Estratégias de Pesquisa

Uma estratégia de pesquisa é definida escolhendo a ordem da expansão do nó.

Para a decisão e avaliação de uma estratégia de pesquisa, foram atendidos os seguintes critérios:

- Completude
- Otimalidade
- Complexidade no tempo
- Complexidade no espaço

Algoritmo	Completo	Otimal	Tempo complexidade	Espaço complexidade
Primeiro em Largura (BFS)	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(b^d)$
Primeiro em Profundidade (DFS)	No	Não	$O(b^m)$	$O(bm)$
Pesquisa Iterativa	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(bd)$
Custo Uniforme	Yes	Sim	Number of nodes with $g(n) \leq C^*$	
Greedy	No	Não	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

Figura 1 - Tabela de classificação dos Algoritmos de Pesquisa.

Pesquisa Não-Informada

As estratégias de pesquisa não informadas usam apenas as informações disponíveis na definição do problema.

Na concretização deste trabalho, apenas foi utilizada o Algoritmo Primeiro em Profundidade (DFS). Para o efeito definem-se as propriedades e características de seguida:

- Estratégia: Expandir sempre um dos nós mais profundos da árvore
- Vantagens: Muito pouco memória necessária, bom para problemas com muitas soluções
- Desvantagens: Não pode ser usada para árvores com profundidade infinita, pode ficar presa em ramos errados

Propriedades:

- Completa: Não, falha em espaços de profundidade infinita, com repetições(*loops*)
- Tempo: $O(b^m)$, mau se $m > d$
- Espaço: $O(bm)$, espaço linear
- Otimal: Não

Legenda:

- b: o máximo fator de ramificação
- d: a profundidade da melhor solução
- m: a máxima profundidade do espaço de estados

Pesquisa Informada

As estratégias de pesquisa informada utilizam informação do problema para evitar que o algoritmo de pesquisa fique “pedido a vaguear no escuro”.

Na concretização deste trabalho, apenas foi utilizada o Algoritmo Pesquisa A*. Para o efeito definem-se as propriedades e características de seguida:

- Estratégia:
 - evitar expandir caminhos que são caros
 - O algoritmo A* combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. Usa a função:

$$f(n) = g(n) + h(n)$$

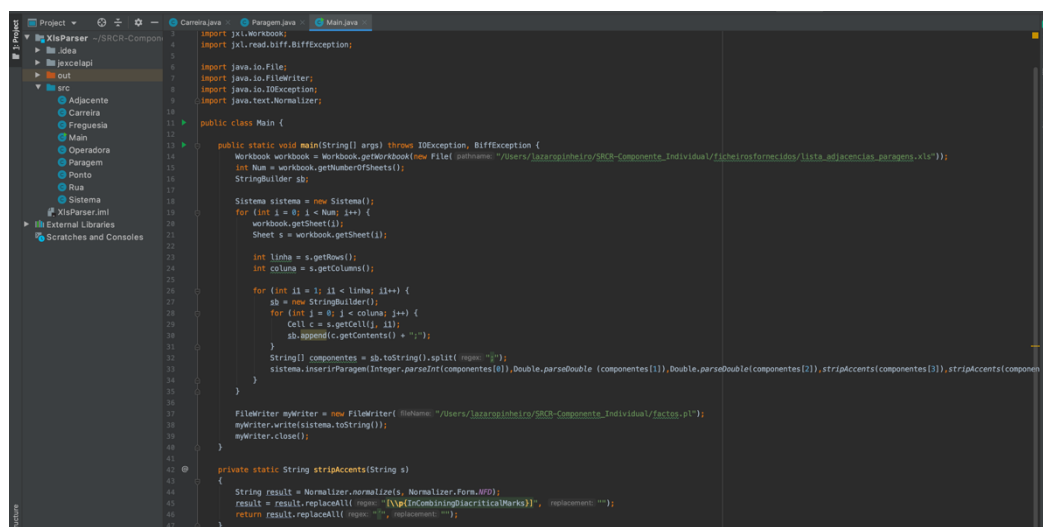
Legenda:

- $g(n)$ - custo total, até agora, para chegar ao estado n (custo do percurso)
- $h(n)$ - custo estimado para chegar ao objetivo
- $f(n)$ - custo estimado da solução mais barata que passa pelo nó n

Resolução das funcionalidades propostas

Obtenção da Base de Conhecimento

Foi desenvolvido um *parser* em Java, de modo a formatar os dados que recebia como input através do ficheiro fornecido (*lista_adjacencias_paragens.xls*), por forma a obter um ficheiro com a extensão *factos.pl*, no qual constam os dados formatados e normalizados para que possam ser processados pelo sistema desenvolvido em PROLOG.



```
1 import jxl.Workbook;
2 import jxl.read.biff.BiffException;
3
4 import java.io.File;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.text.Normalizer;
8
9 public class Main {
10
11     public static void main(String[] args) throws IOException, BiffException {
12         Workbook workbook = Workbook.getWorkbook(new File("D:/Users/lazarpinheiro/SRCR-Componente_Individual/ficheirosfornecidos/lista_adjacencias_paragens.xls"));
13         int Num = workbook.getNumberOfSheets();
14         StringBuilder sb;
15
16         Sistema sistema = new Sistema();
17         for (int i = 0; i < Num; i++) {
18             Workbook sheet = workbook.getSheet(i);
19             Sheet s = workbook.getSheet(i);
20
21             int linha = s.getRow(1);
22             int coluna = s.getColumns(1);
23
24             for (int il = 1; il < linha; il++) {
25                 sb = new StringBuilder();
26                 for (int j = 0; j < coluna; j++) {
27                     Cell c = s.getCell(j, il);
28                     sb.append(c.getContents() + ";");
29                 }
30                 String[] componentes = sb.toString().split(";");
31                 sistema.inserirParagem(Integer.parseInt(componentes[0]), Double.parseDouble(componentes[1]), Double.parseDouble(componentes[2]), stripAccents(componentes[3]), stripAccents(componentes[4]));
32             }
33         }
34
35         FileWriter myWriter = new FileWriter("D:/Users/lazarpinheiro/SRCR-Componente_Individual/factos.pl");
36         myWriter.write(sistema.toString());
37         myWriter.close();
38     }
39
40     private static String stripAccents(String s) {
41         String result = Normalizer.normalize(s, Normalizer.Form.NFD);
42         result = result.replaceAll("\\p{InCombiningDiacriticalMarks}", "");
43         return result.replaceAll("\\p{Mn}", "");
44     }
45 }
```

Figura 2 - Main do Parser realizado em Java.

Base de Conhecimento

Apresenta-se de seguida a formatação dos factos que constam na base de conhecimento.

```
% FREGUESIAS
% Extensão do predicado freguesia: IdFreguesia, Nome -> {V,F}

freguesia( 1, 'Alges, Linda-a-Velha e Cruz Quebrada-Dafundo' ).
freguesia( 0, 'Carnaxide e Queijas' ).
freguesia( 3, 'Porto Salvo' ).
freguesia( 2, 'Barcarena' ).
freguesia( 4, 'Oeiras e Sao Juliao da Barra, Paco de Arcos e Caxias' ).
```

Figura 3 - Formatação do predicado representativo do facto freguesia.

```
% RUAS
% Extensão do predicado rua: Codigo, Nome, IdFreguesia -> {V,F}

rua( 10, 'Avenida dos Bombeiros Voluntarios de Alges', 1 ).
rua( 25, 'Estrada de Alfragide', 0 ).
rua( 51, 'Rua A Gazeta DOeiras', 4 ).
rua( 70, 'Rua Actor Antonio Pinheiro', 3 ).
rua( 79, 'Rua dos Acores', 0 ).
rua( 83, 'Rua Angra do Heroismo', 0 ).
rua( 102, 'Largo Dom Manuel I', 1 ).
```

Figura 4 - Formatação do predicado representativo do facto rua.

```
% OPERADORAS
% Extensão do predicado operadora: Operadora, Nome -> {V,F}

operadora( 0, 'Vimeca' ).
operadora( 1, 'SCoTTURB' ).
operadora( 2, 'LT' ).
operadora( 3, 'Carris' ).
```

Figura 5 - Formatação do predicado representativo do facto operadora.

```
% PARAGENS
% Extensão do predicado paragem: Gid, Latitude, Longitude, Estado, Tipo, Publicidade, Carreiras, Operadora, Codigo -> {V,F}

paragem( 5, -106997.31, -95311.49, 'Bom', 'Sem Abrigo', 'No', [ 776 ], 3, 103 ).
paragem( 6, -106992.24, -95299.38, 'Bom', 'Sem Abrigo', 'No', [ 6 ], 0, 103 ).
paragem( 8, -106980.35, -95289.3, 'Bom', 'Sem Abrigo', 'No', [ 776 ], 3, 103 ).
paragem( 9, -107003.0, -95216.21, 'Bom', 'Fechado dos Lados', 'Yes', [ 6 ], 0, 103 ).
paragem( 10, -107129.12, -102327.55, 'Bom', 'Fechado dos Lados', 'Yes', [ 106, 112, 122 ], 2, 545 ).
paragem( 11, -105158.82133137222, -95894.13861202101, 'Bom', 'Fechado dos Lados', 'Yes', [ 201, 748, 751 ], 3, 416 ).
paragem( 12, -105655.76, -95028.52, 'Bom', 'Fechado dos Lados', 'Yes', [ 162 ], 2, 116 ).
```

Figura 4 - Formatação do predicado representativo do facto paragem.

```

1065 % ADJACENTES
1066 % Extensão do predicado adjacente: GidOrigem, GidDestino, Carreira, Distancia -> {V,F}
1067
1068 adjacente( 183, 791, 1, 87.63541293336934 ).
1069 adjacente( 791, 595, 1, 698.6929317661744 ).
1070 adjacente( 595, 182, 1, 421.9863463431075 ).
1071 adjacente( 182, 499, 1, 2003.3340491291042 ).
1072 adjacente( 499, 593, 1, 245.01549440800457 ).
1073 adjacente( 593, 181, 1, 1734.5336738731814 ).
1074 adjacente( 181, 180, 1, 127.2539885425926 ).

```

Figura 5 - Formatação do predicado representativo do facto adjacente.

Desenvolvimento de Requisitos

Seguidamente apresentam-se os trechos de código que satisfazem os requisitos propostos. A legenda de cada figura indica qual o requisito solicitado e o respetivo algoritmo usado na pesquisa.

```

% -----
% Calcular um trajeto entre dois pontos.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados).
% -----

trajetoEntrePontos(Origem, Destino, Caminho) :- calculaTrajeto(Origem, Destino, [Origem], Caminho).

calculaTrajeto(Destino, Destino, Visitadas, Caminho) :- inverso(Visitadas, Caminho).
calculaTrajeto(Origem, Destino, Visitadas, Caminho) :- isAdjacente(Origem, Proxima),
\+ member(Proxima, Visitadas),
calculaTrajeto(Proxima, Destino, [Proxima|Visitadas], Caminho).

```

Figura 8 - Calcular um trajeto entre dois pontos (DFS).

```

% -----
% Selecionar apenas algumas das operadoras de transporte para um determinado percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoEntrePontosPorOperadora(Origem, Destino, Operadoras, Caminho) :- calculaTrajetoPorOperadora(Origem, Destino, Operadoras, [Origem], Caminho).

calculaTrajetoPorOperadora(Destino, Destino, Operadoras, Visitadas, Caminho) :- paragem( Destino, _ , _ , _ , _ , OperadoraDestino, _ ),
member(OperadoraDestino, Operadoras),
inverso(Visitadas, Caminho).
calculaTrajetoPorOperadora(Origem, Destino, Operadoras, Visitadas, Caminho) :- paragem( Origem, _ , _ , _ , _ , OperadoraOrigem, _ ),
member(OperadoraOrigem, Operadoras),
isAdjacente(Origem, Proxima),
\+ member(Proxima, Visitadas),
calculaTrajetoPorOperadora(Proxima, Destino, Operadoras, [Proxima|Visitadas], Caminho).

```

Figura 9 - Selecionar apenas algumas das operadoras de transporte para um determinado percurso (DFS).

```

% -----
% Excluir um ou mais operadores de transporte para o percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoEntrePontosSemOperadora(Origem, Destino, Operadoras, Caminho) :- calculaTrajetoSemOperadora(Origem, Destino, Operadoras, [Origem], Caminho).

calculaTrajetoSemOperadora(Destino, Destino, Operadoras, Visitadas, Caminho) :- paragem( Destino, _ , _ , _ , _ , OperadoraDestino, _ ),
\+ member(OperadoraDestino, Operadoras),
inverso(Visitadas, Caminho).
calculaTrajetoSemOperadora(Origem, Destino, Operadoras, Visitadas, Caminho) :- paragem( Origem, _ , _ , _ , _ , OperadoraOrigem, _ ),
\+ member(OperadoraOrigem, Operadoras),
isAdjacente(Origem, Proxima),
\+ member(Proxima, Visitadas),
calculaTrajetoSemOperadora(Proxima, Destino, Operadoras, [Proxima|Visitadas], Caminho).

```

Figura 6 - Excluir um ou mais operadores de transporte para o percurso (DFS).

```

% -----
% Identificar quais as paragens com o maior número de carreiras num determinado percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoParagensComMaisCarreiras(Origem, Destino, Caminho/Carreiras) :- paragem( Origem, _, _, _, CarreirasOrigem, _ ),
length(CarreirasOrigem, NumCarreirasOrigem),
calculaParagensComMaisCarreiras(Origem, Destino, [Origem], [Origem]/NumCarreirasOrigem, Caminho/Carreiras).

calculaParagensComMaisCarreiras(Destino, Destino, Visitadas, MaxCarreiras, Caminho/Carreiras) :- paragem( Destino, _, _, _, CarreirasDestino, _ ),
length(CarreirasDestino, NumCarreirasDestino),
paragemComMaisCarreiras(Destino/NumCarreirasDestino, MaxCarreiras, Carreiras),
inverso(Visitadas, Caminho).

calculaParagensComMaisCarreiras(Origem, Destino, Visitadas, MaxCarreiras, Caminho/Carreiras) :- isAdjacente(Origem, Proxima),
\+ member(Proxima, Visitadas),
paragem( Proxima, _, _, _, CarreirasProxima, _ ),
length(CarreirasProxima, NumCarreirasProxima),
paragemComMaisCarreiras(Proxima/NumCarreirasProxima, MaxCarreiras, Result),
calculaParagensComMaisCarreiras(Proxima, Destino, [Proxima|Visitadas], Result, Caminho/Carreiras).

```

Figura 11- Identificar quais as paragens com o maior número de carreiras num determinado percurso (DFS).

```

% -----
% Escolher o menor percurso (usando critério menor número de paragens).
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

percursoMenosParagens(Origem, Destino, Caminho/NP):- findall( (S,NumParagens),
( trajetoEntrePontos(Origem, Destino, S),length(S, NumParagens) ),
Lista ), minimo(Lista, Caminho/NP).

```

Figura 12 - Escolher o menor percurso (usando o critério menor número de paragens) (DFS).

```

% -----
% Escolher o percurso mais rápido (usando critério da distância).
% Algoritmo de Pesquisa : Pesquisa Informada( Algoritmo A* )
% -----

percursoMaisRapido_Distancia(Origem, Destino, Caminho/Custo) :-
distanciaEuclidiana(Origem, Destino, Estima),
calculaMaisRapido_Distancia(Destino, [[Origem]/0/Estima], InvCaminho/Custo/_),
inverso(InvCaminho, Caminho).

calculaMaisRapido_Distancia(Destino, Caminhos, Caminho) :-
obtem_MaisRapido_Distancia(Destino, Caminhos, Caminho),
Caminho = [Destino|_]/_/_.

calculaMaisRapido_Distancia(Destino, Caminhos, SolucaoCaminho) :-
obtem_MaisRapido_Distancia(Destino, Caminhos, MelhorCaminho),
seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
expande_MaisRapido_Distancia(Destino, MelhorCaminho, ExpCaminhos),
append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
calculaMaisRapido_Distancia(Destino, NovoCaminhos, SolucaoCaminho).

obtem_MaisRapido_Distancia(Destino, [Caminho], Caminho) :- !.

obtem_MaisRapido_Distancia(Destino, [Caminho1/Custo1/Est1, _/Custo2/Est2|Caminhos], MelhorCaminho) :-
Custo1 + Est1 <= Custo2 + Est2, !,
obtem_MaisRapido_Distancia(Destino, [Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).

obtem_MaisRapido_Distancia(Destino, [_|Caminhos], MelhorCaminho) :-
obtem_MaisRapido_Distancia(Destino, Caminhos, MelhorCaminho).

expande_MaisRapido_Distancia(Destino, Caminho, ExpCaminhos) :-
findall(NovoCaminho, isAdjacente(Destino, Caminho, NovoCaminho), ExpCaminhos).

isAdjacente(Destino, [Origem|Caminho]/Custo/_, [Proxima, Origem|Caminho]/NovoCusto/Estima) :-
adjacente(Origem, Proxima, _, PassoCusto),
\+ member(Proxima, Caminho),
NovoCusto is Custo + PassoCusto,
distanciaEuclidiana(Proxima, Destino, Estima).

```

Figura 137 - Escolher o percurso mais rápido (usando critério da distância) (A*).

```

% -----
% Escolher o percurso que passe apenas por abrigos com publicidade.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoEntrePontosComPublicidade(Origem, Destino, Caminho) :- calculaTrajetoComPublicidade(Origem, Destino, [Origem], Caminho).

calculaTrajetoComPublicidade(Destino, Destino, Visitadas, Caminho) :- paragem( Destino, _ , _ , _ , 'Yes', _ , _ , _ ), inverso(Visitadas, Caminho).
calculaTrajetoComPublicidade(Origem, Destino, Visitadas, Caminho) :- isAdjacente(Origem, Proxima),
                                                                    paragem( Origem, _ , _ , _ , 'Yes' , _ , _ , _ ),
                                                                    \+ member(Proxima, Visitadas),
                                                                    calculaTrajetoComPublicidade(Proxima, Destino, [Proxima|Visitadas], Caminho).

```

Figura 8 - Escolher o percurso que passe apenas por abrigos com publicidade (DFS).

```

% -----
% Escolher o percurso que passe apenas por paragens abrigadas.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoEntrePontosComAbrigo(Origem, Destino, Caminho) :- paragem( Destino, _ , _ , _ , Abrigo, _ , _ , _ ),
                                                                    \+ member(Abrigo, ['Sem Abrigo']),
                                                                    calculaTrajetoComAbrigo(Origem, Destino, [Origem], Caminho).

calculaTrajetoComAbrigo(Destino, Destino, Visitadas, Caminho) :- inverso(Visitadas, Caminho).
calculaTrajetoComAbrigo(Origem, Destino, Visitadas, Caminho) :- isAdjacente(Origem, Proxima),
                                                                    paragem( Origem, _ , _ , _ , Abrigo, _ , _ , _ ),
                                                                    \+ member(Abrigo, ['Sem Abrigo']),
                                                                    \+ member(Proxima, Visitadas),
                                                                    calculaTrajetoComAbrigo(Proxima, Destino, [Proxima|Visitadas], Caminho).

```

Figura 9 - Escolher o percurso que passe apenas por paragens abrigadas (DFS).

```

% -----
% Escolher um ou mais pontos intermédios por onde o percurso deverá passar.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----

trajetoPorIntermedias(Origem, Destino, Intermedias, Caminho) :- removeEmComum(Origem, Intermedias, NovasIntermedias),
                                                                    removeEmComum(Origem, Intermedias, NovasIntermedias),
                                                                    calculaTrajetoPorIntermedias(Origem, Destino, NovasIntermedias, [Origem], Caminho).

calculaTrajetoPorIntermedias(Destino, Destino, [], Visitadas, Caminho) :- inverso(Visitadas, Caminho).
calculaTrajetoPorIntermedias(Origem, Destino, Intermedias, Visitadas, Caminho) :- isAdjacente(Origem, Proxima),
                                                                    \+ member(Proxima, Visitadas),
                                                                    (member(Proxima, Intermedias) => removeEmComum(Proxima, Intermedias, NovasIntermedias) ; NovasIntermedias = Intermedias),
                                                                    calculaTrajetoPorIntermedias(Proxima, Destino, NovasIntermedias, [Proxima|Visitadas], Caminho).

```

Figura 10 - Escolher um ou mais pontos intermédios por onde o percurso deverá passar (DFS).

Criação de uma API de modo a manipular as pesquisas

Foi criada uma API de modo a facilitar a interação do utilizador com o sistema, onde está acede aos algoritmos de pesquisa para cada requisito.

```

% -----
% Calcular um trajeto entre dois pontos.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados).
% -----

trajetoEntrePontos(Origem, Destino) :- trajetoEntrePontos(Origem, Destino, Caminho),
                                                                    write('-----\n'),
                                                                    write('----- Trajeto Entre Pontos ----- \n'),
                                                                    write('-----\n'),
                                                                    write('\nOrigem: '),
                                                                    write(Origem),
                                                                    write('\nDestino: '),
                                                                    write(Destino),
                                                                    write('\nPercurso: '),
                                                                    write(Caminho).

```

Figura 11 – Definição da interface com o utilizador para o requisito “Calcular um trajeto entre dois pontos”.

```

% -----
% Selecionar apenas algumas das operadoras de transporte para um determinado percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoEntrePontosPorOperadora(Origem, Destino, Operadoras) :- parseOperadoras(Operadoras, OperadorasParsed),
trajetoEntrePontosPorOperadora(Origem, Destino, OperadorasParsed, Caminho),
write('-----\n'),
write('----- Trajeto Entre Pontos Por Operadoras ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nOperadoras: '),
write(Operadoras),
write('\nPercurso: '),
write(Caminho).

```

Figura 12 – Definição da interface com o utilizador para o requisito “Selecionar apenas algumas das operadoras de transporte para um determinado percurso”.

```

% -----
% Excluir um ou mais operadores de transporte para o percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoEntrePontosSemOperadora(Origem, Destino, Operadoras) :- parseOperadoras(Operadoras, OperadorasParsed),
trajetoEntrePontosSemOperadora(Origem, Destino, OperadorasParsed, Caminho),
write('-----\n'),
write('----- Trajeto Entre Pontos Sem Operadoras ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nOperadoras(Nao Permitidas): '),
write(Operadoras),
write('\nPercurso: '),
write(Caminho).

```

Figura 13 – Definição da interface com o utilizador para o requisito “Excluir um ou mais operadores de transporte para o percurso”.

```

% -----
% Identificar quais as paragens com o maior número de carreiras num determinado percurso.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoParagensComMaisCarreiras(Origem, Destino) :- trajetoParagensComMaisCarreiras(Origem, Destino, Caminho/(Paragens/NumParagens)),
write('-----\n'),
write('----- Trajeto Paragens Com Mais Carreiras ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nParagens com mais carreiras:'),
write(Paragens),
write('\nNumero de Paragens: '),
write(NumParagens),
write('\nPercurso: '),
write(Caminho).

```

Figura 14 – Definição da interface com o utilizador para o requisito “Identificar quais as paragens com o maior número de carreiras num determinado percurso”.


```

% -----
% Escolher o menor percurso (usando critério menor número de paragens).
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
percursoMenosParagens(Origem, Destino) :- percursoMenosParagens(Origem, Destino, Caminho/NumParagens),
write('-----\n'),
write('----- Trajeto Com Menos Paragens ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nNúmero de Paragens: '),
write(NumParagens),
write('\nPercurso: '),
write(Caminho).

```

Figura 15 - Definição da interface com o utilizador para o requisito “Escolher o menor percurso (usando o critério menor número de paragens)”.

```

% -----
% Escolher o percurso mais rápido (usando critério da distância).
% Algoritmo de Pesquisa : Pesquisa Informada( Algoritmo A* )
% -----
percursoMaisRapido_Distancia(Origem, Destino) :- percursoMaisRapido_Distancia(Origem, Destino, Caminho/Custo),
write('-----\n'),
write('----- Trajeto Mais Rapido (Distancia) ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nPercurso: '),
write(Caminho),
write('\nDistancia Total: '),
write(Custo).

```

Figura 16 - Definição da interface com o utilizador para o requisito “Escolher o percurso mais rápido (usando critério da distância)”.

```

% -----
% Escolher o percurso que passe apenas por abrigos com publicidade.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoEntrePontosComPublicidade(Origem, Destino) :- trajetoEntrePontosComPublicidade(Origem, Destino, Caminho),
write('-----\n'),
write('----- Trajeto Com Publicidade ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nPercurso: '),
write(Caminho).

```

Figura 17 - Definição da interface com o utilizador para o requisito “Escolher o percurso que passe apenas por abrigos com publicidade”.


```
% -----
% Escolher o percurso que passe apenas por paragens abrigadas.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoEntrePontosComAbrigo(Origem, Destino) :- trajetoEntrePontosComAbrigo(Origem, Destino, Caminho),
write('-----\n'),
write('----- Trajeto Com Abrigo ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nPercurso: '),
write(Caminho).
```

Figura 18 - Definição da interface com o utilizador para o requisito “Escolher o percurso que passe apenas por paragens abrigadas”.

```
% -----
% Escolher um ou mais pontos intermédios por onde o percurso deverá passar.
% Algoritmo de Pesquisa : Pesquisa Não-Informada( Primeiro em Profundidade Com Multi-Estados)
% -----
trajetoPorIntermedias(Origem, Destino, Intermedias) :- trajetoPorIntermedias(Origem, Destino, Intermedias, Caminho),
write('-----\n'),
write('----- Trajeto Com Abrigo ----- \n'),
write('-----\n'),
write('\nOrigem: '),
write(Origem),
write('\nDestino: '),
write(Destino),
write('\nParagens Intermedias: '),
write(Intermedias),
write('\nPercurso: '),
write(Caminho).
```

Figura 19 - Definição da interface com o utilizador para o requisito “Escolher um ou mais pontos intermédios por onde o percurso deverá passar”.

Execução da Aplicação

A imagem ilustra a interface gráfica que é apresentada ao utilizador da aplicação. Desta forma, a interação deste dois elementos é mais fácil, visto que tudo está organizado. Quando o resultado é uma lista demasiado extensa, o sistema imprime para o *stdout* toda a informação, ultrapassando a limitação apresentada pelo *SICSTUS*.

```
| ?- percursoMenosParagens(670,667).
```

```
-----
----- Trajeto Com Menos Paragens -----
-----

Origem: 670
Destino: 667
Numero de Paragens: 3
Percurso: [670,676,667]
yes
```

Figura 20 - Exemplo de Interface com o Utilizador

Conclusões e sugestões

Concluída a execução do trabalho importa salientar que este permitiu uma maior consolidação e aprofundamento acerca da linguagem de programação PROLOG e a utilização da mesma em estratégias de pesquisas sobre Grafos.

No decurso deste trabalho, alcançar os predicados adequados para a base de conhecimento foi um enorme desafio, dado ser uma das partes fulcral do trabalho, pois as pesquisas socorrem-se desta estrutura. Por outro lado, foi complexo encontrar os algoritmos de pesquisas adequados para pesquisar no Grafo.

O enunciado, apesar de toda a análise cuidadosa e profunda para uma boa interpretação, não foi difícil interpretar e aplicar à situação apresentada.

Os objetivos foram, portanto, alcançados com sucesso, dando resposta a todos os requisitos presentes no enunciado, apesar de ser evidente o limite de tempo para a concretização.

Futuramente, seria vantajoso, além de um período mais prolongado de tempo, alargar este trabalho com um maior número de algoritmos de pesquisa, procurando melhorar a performance aumentando os algoritmos.

Referências

Paragens Autocarro OEIRAS VALLEY. Disponível em:
<http://dadosabertos.cm-oeiras.pt/dataset/paragens-de-autocarro>

[Russel, 2009] RUSSEL, Stuart, NORVIG, Peter,
“Artificial Intelligence – A Modern Approach, 3rd edition”.

[Bratko, 2000] BRATKO, Ivan,
"PROLOG: Programming for Artificial Intelligence", 3rd Edition,
Addison-Wesley Longman Publishing Co., Inc., 2000