UNIVERSIDADE DO MINHO

SISTEMA DE TRANSPORTES DO CONCELHO DE OEIRAS

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Sistemas de Representação de Conhecimento e Raciocínio

(2.º Semestre / 2019-20)

A84727 Nelson Faria

Braga
5 de Junho de 2020

Resumo

Neste trabalho pretendeu-se desenvolver um sistema de resolução de Problemas e de Procura a partir de *datasets* relativos às paragens de autocarros do concelho de Oeiras. Para isso desenvolveu-se várias técnicas de Pesquisa, tanto **Pesquisa não Informada**(como, por exemplo, a *Pesquisa Breadth First*) como **Pesquisa Informada**(como, por exemplo, a *Pesquisa A**). O principal foco do trabalho foi o uso da linguagem de programação *PROLOG* para materializar os conhecimentos aprendidos na cadeira de **Sistemas de Representação de Conhecimento e Raciocínio** ao longo do Semestre.

Conteúdo

Capítulo 1

Introdução

Neste trabalho de Avaliação Individual o que se pretendia era desenvolver um sistema que, a partir de métodos de Resolução de Problemas e de Procura, fosse capaz de recomendar ao utilizador, mediante aquilo que ele pretende, diversas formas de se mover no concelho de Oeiras a partir de transportes Públicos.

Para isso, foi essencial importar os dados, que estavam presentes nos *datasets* das paragens e das adjacências entre paragens, para o *PROLOG*. Depois dos dados estarem todos importados, a partir daí é que se desenvolveu o caso prático em estudo, nomeadamente dar resposta às pesquisas pretendidas no enunciado deste Trabalho Prático, mas também responder a outras questões que porventura possam ser interessantes para o caso.

Assim, de modo geral, o principal objetivo foi desenvolver uma Base de Conhecimento em *PROLOG* que implementasse algumas das matérias que foram lecionadas ao longo do semestre em **SRCR**, principalmente algumas das estratégias de Pesquisa.

Capítulo 2

Preliminares

Ao longo do semestre foram lecionadas várias estratégias de pesquisa de maneira a se resolverem problemas de Pesquisa, onde basicamente o que se faz é reduzir um problema que pode ser visto como um grafo numa árvore de Pesquisa.

Os tipos de Pesquisa que existem são: **Pesquisa Informada** e **Pesquisa Não Informada**, sendo duas abordagens bastante distintas na maneira como abordam o problema. A **Pesquisa Não Informada** (ou pesquisa 'cega') o que faz é somente utilizar a informação que foi definida no início aquando da definição do problema, sendo neste caso as informações das paragens e das adjacências entre paragens. Assim, este tipo de Pesquisa, por vezes, torna-se muito ineficiente quando se lida com uma quantidade tão grande de dados, como é o caso. Assim, as várias estratégias de pesquisa não informada são: Pesquisa em profundidade, Pesquisa em Largura, Pesquisa de custo uniforme, Pesquisa Iterativa e Pesquisa Bidirecional.

Ao nível da **Pesquisa Não Informada**, neste trabalho só foram usadas as estratégias de Pesquisa em profundidade e em largura, sendo que a pesquisa em profundidade significa basicamente que são procurados primeiro os ramos até às folhas da árvore, de maneira que para isso não é necessário haver o guardar dos nodos que ainda faltam visitar, pois este tipo de método continua recursivamente até chegar a uma folha da árvore, só depois é que passa para outro ramo da árvore. Os problemas desta abordagem é que pode ficar presa em sítios da árvore que não devia, ou até entrar em ciclo infinito. Quanto à pesquisa em largura, o que acontece é um pouco diferente, pois aqui são guardados os dados dos nodos adjacentes a um determinado nodo raiz, pelo que esta pesquisa vai alargando a orla (o número de nodos que faltam visitar) até que chega a encontrar o destino, ou então já visitou todos os nodos, não havendo mais nenhum para visitar. O problema desta abordagem é que necessita de mais memória do que a pesquisa em profundidade, o que pode dar problemas para problemas complexos com um elevado número de nodos. Por fim, ambas as estratégias de pesquisa em profundidade e em largura não chegam imediatamente à solução ótima, pois basicamente a primeira que eles encontram pode não ser a melhor de todas.

Ao nível da **Pesquisa Informada**, algumas das estratégias vistas nas aulas foram a **Pesquisa A*** e a **Pesquisa Gulosa(*Greedy*)**. A Pesquisa Gulosa consiste em expandir somente os nodos que parecem estar mais próximos do destino esperado, pelo que para isso ela precisa somente de uma estimativa para que consiga estimar a melhor solução a partir de um determinado nodo. Aquele que possuir a melhor estimativa, de acordo com a pesquisa gulosa, encontra-se mais perto da solução. O problema desta abordagem é que não tem em conta o trajeto que foi efetuado até ao momento, pelo que nem sempre dá logo a solução ótima, sendo por isso que existe a pesquisa A*. Na pesquisa A*, o que acontece é que para além daquilo que acontece na pesquisa gulosa, ainda se têm em conta o trajeto efetuado até ao momento aquando da decisão de qual nodo seguir na pesquisa. Porém todas estas duas abordagens

necessitam de boas estimativas para que consigam dar os melhores resultados possíveis, pelo que se a estimativa for boa, a Pesquisa A* pode dar logo a solução ótima.

Por último, existem outras partes importantes que também foram usadas neste trabalho e que foram muito importantes no trabalho de Grupo, que foram o *Pressuposto do Mundo Fechado* e *Programação em Lógica Estendida*, pois, por exemplo, nos *datasets* podem existir informações que não são conhecidas, mas a sua inexistência não é importante para o objetivo do trabalho, como por exemplo informações relativas às freguesias das paragens.

Capítulo 3

Descrição do Trabalho e Análise de Resultados

3.1 Breve Descrição do problema

Neste problema o que se pretendia era representar os dados contidos num *dataset* em *Excel* no *PROLOG* e depois realizar operações de Pesquisa sobre eles. Para isso, comecei por ler os dados do Excel e depois construir os predicados respetivos, sendo nesta primeira fase do trabalho importante a construção de um *Parser* que me dividisse os dados em campos e depois os escreve-se num ficheiro de forma a constituírem a base de Conhecimento do sistema de transportes públicos do concelho de Oeiras. Todo o *Parser* que fiz foi construído usando a linguagem de Programação *JAVA* e a sua construção será detalhada numa das secções seguintes.

3.2 Tratamento dos dados dos *Datasets*

Os dados do *dataset* "Paragens de Autocarro do Concelho de Oeiras"estão organizados da seguinte forma:

1. Latitude
2. Longitude
3. Gid (identificador da paragem)
4. Estado de Conservação
5. Tipo de Abrigo
6. Abrigo com Publicidade
7. Operadora
8. Carreiras
9. Código de Rua
10. Nome da Rua
11. Freguesia

Deste modo, a partir destes dados, construíu-se uma classe em *JAVA* que lê todas as linhas do ficheiro *Excel* e depois as guarda-se em objetos representativos das Paragens, sendo que os atríbutos desses objetos são os que estão mencionados acima. Já no caso do *dataset* "Listas Adjacencias Paragens", foi importante ter em conta que os dados encontravam-se organizados por folha, sendo cada folha do Excel representativa de uma Carreira. Assim, o que foi feito foi guardar a informação presente no *dataset*, nomeadamente guardar os objetos das paragens presentes nesse ficheiro num objeto que representa a carreira e depois guardar as carreiras, logo após a leitura de uma folha com todas as paragens presentes numa determinada carreira. É importante notar que os dados que estavam presentes nos ficheiros *Excel* não podiam ser introduzidos integralmente como estavam no *Excel*, uma vez que poderiam conter espaços ou até vírgulas e letras maiúsculas, pelo que desta forma não é possível introduzir a informação no *PROLOG* de forma a não produzir erros. Assim, antes de se guardarem as Strings o que se fez foi prepara-las de maneira a não darem erros no *PROLOG*.

Outro aspeto importante que se teve em conta foi os predicados que não tinham certas informações que não eram de todos importantes para o funcionamento correto do problema. Por exemplo, a base de Conhecimento pode ter uma paragem onde não se conhece o nome da Freguesia ou até o nome da Rua. Deste modo, aquando da leitura reconheceu-se estes casos e adicionou-se uma pequena marca única para cada caso de forma a se poder depois, na fase de escrita, identificarem-se estes casos e tratar deles. Basicamente na fase de escrita depois o que se faz é identificar estes casos e construír os predicados com essa informação, pelo que estes tipos de valores foram encarados como valores desconhecidos do tipo I. Deste modo, no *JAVA*, ainda houve necessidade de se construírem predicados para se lidarem com estas exceções. De seguida mostra-se um exemplo destes casos:

```
paragem( 752,-103260.70410270982,-101287.68122082386,bom,xptoSRCR0,no,lt,[106,
119],262.0,'estrada__de__leceia','porto__salvo' ).

excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Oper,Carr,Cod,Nome,Freg ) ) :-
      paragem( Gid,Lat,Long,Estado,xptoSRCR0,Pub,Oper,Carr,Cod,Nome,Freg ).
```

Para ler dos dois ficheiros *Excel* foi necessário usar uma biblioteca *JAVA* denominada *Apache POI (versão 4.1.2)* de maneira a conseguir-se retirar toda as informações contidas tanto nas várias linhas e colunas, mas também nas várias páginas que o ficheiro possa ter.

Depois de toda a informação guardada em objetos, procedeu-se à fase de Escrita dos dados em ficheiro. Para isso, foi importante definir uma estrutura para os predicados, de maneira a representar a informação de forma mais eficiente e funcional possível. A estrutura dos predicados que foi definida é a seguinte:

- **paragem:** #IdParag, Latitude, Longitude, Estado,Tipo de Abrigo, Abrigo com Publicidade, Operadora, Carreiras, Código de Rua, Nome da Rua, Freguesia $\hookrightarrow\{V, F\}$
- **adjacencia:** #Carreira, #IdParagA, #IdParagB, Distancia $\hookrightarrow\{V, F\}$

Deste modo, e usando as funcinalidades do *JAVA*, criou-se então dois métodos que foram responsáveis pela escrita do ficheiro que contém todos os predicados das paragens e o ficheiro que contém todos os predicados das adjacencias.

De referir ainda que no ficheiro das adjacências só foram guardados, no predicado respetivo, o número da Carreira a que se refere a adjacência e os ids das duas paragens envolvidas numa adjacência, pois o outro ficheiro já contém a informação das paragens. Contudo , é importante salientar que existe mais um parâmetro nas adjacências que é a distância euclidiana entre as duas paragens. Isto foi feito aquando da escrita no ficheiro para otimizar depois no *PROLOG*

as pesquisas, dado que o custo de deslocação entre duas paragens consecutivas será sempre representado por essa distância, e ela é sempre a mesma. Outro aspeto importante é que o grafo constituído por este problema é unidirecional, no meu caso, porque achei que faria mais sentido que uma carreira fosse somente percorrida num sentido, se fosse nos dois teria de ter um outro id respetivo a outra carreira diferente, pois é o que acontece na realidade.

Depois de gerados os ficheiros, no ficheiro da base de Conhecimento das Paragens teve ainda de se fazer uns *finds/replaces* de forma a tirar os caractéres acentuados do ficheiro porque senão quando uma palavra tinha um certo caractér com acentos ela aparecia no *SICStus* com um encoding esquisito.

Todo o código relativo a esta parte do *Parser* em *JAVA* está presente numa secção abaixo dos Anexos.

3.3 Análise dos Dados das Bases de Conhecimento

Depois de se concluir a fase de leitura e escrita dos dados dos *datasets* das Paragens do concelho de Oeiras, é altura de passar para o *PROLOG*. Aqui basicamente o que se fez em primeiro lugar foi importar os dados dos dois ficheiros com o conhecimento das paragens e das adjacências através do uso das primitivas : $-include('paragensBC.prolog.pl')$. e : $-include('adjacenciasBC.prolog.pl')$., sendo o que está dentro de parênteses o nome dos ficheiros respetivos. Depois, ainda se aplicou o PMF (*Pressuposto do Mundo Fechado*) para os predicados *paragem* e *adjacencia* conforme se mostra a seguir:

```
-paragem( Gid,Lat,Long,Estado,Tipo,Pub,Oper,Carr,Cod,Nome,Freg ) :-
    nao(paragem( Gid,Lat,Long,Estado,Tipo,Pub,Oper,Carr,Cod,Nome,Freg )),
    nao(excecao(paragem( Gid,Lat,Long,Estado,Tipo,Pub,Oper,Carr,Cod,Nome,Freg ))).

-adjacencia( Carreira,ParagemA,ParagemB,Dist ) :-
     nao(adjacencia( Carreira,ParagemA,ParagemB,Dist )),
     nao(excecao(adjacencia( Carreira,ParagemA,ParagemB,Dist ))).
```

Além disso, foram ainda construídos dois invariantes, pois o conhecimento, apesar de poder ser estendido, não pode ser diminuído, pelo que os invariantes que foram feitos servem para não permitir remoções da base de Conhecimento..

Seguidamente, o que se fez foi implementar na realidade as funcionalidades pretendidas, nomeadamente a aplicação dos princípios de pesquisa aprendidos durante o semestre, as consultas pedidas no enunciado e ainda algumas funcionalidades extra.

3.3.1 Pesquisas Não Informadas

As pesquisas não informadas que foram utilizadas foram as pesquisas em profundidade e em largura, como já referido anteriormente. De referir que um caminho neste exemplo têm de ter em conta que existem várias paragens com mais do que uma carreira a lá passar, pelo que a um nodo está sempre associado a sua informação relativa à carreira que lhe deu origem.

• Pesquisa em Profundidade

Deste modo, começou-se por fazer um predicado para a pesquisa em profundidade (**Depth-First Search**) sem custos envolvidos. Neste caso, e sabendo os pressupostos da pesquisa em profundidade que já foram descritos na secção Preliminares 2, dada uma origem e um destino, o que é devolvido é as paragens e as carreiras percorridas nesse percurso. De notar que era

importante possuir um histórico que me guardasse todos os nodos que já forma visitados de modo a não deixar que ocorram ciclos. Contudo, penso que devido ao *backtracking* existente no *PROLOG*, o que acontece é que ao enviar o histórico para os nodos folha, quando chega a um nodo terminal há o *backtracking*, passando o histórico a ficar sem as ocorrências dos nodos por onde passou nessa profundidade que ele atingiu. Assim, devido à complexidade dos dados, para certos caminhos o programa entra em ciclo, o que só poderia ser resolvido, na minha opinião, ou com um critério de paragem que detetasse a existência de ciclo, ou então que todo o histórico fosse guardado de tal forma que depois de haver *backtracking* os dados dos nodos visitados antes nessa profundidade fossem todos guardados, não permitindo no futuro que se volte a visitá-los.

A minha resolução para este problema mais genérica é:

```
adjacente( Carreira,Nodo,ProxNodo ) :-
        adjacencia( Carreira,Nodo,ProxNodo,_ ).


percurso_pp( Origem,Destino,[Origem|Caminho],Carreiras ) :-
        existeParagem( Origem ), existeParagem( Destino ),
        profundidadeprimeiro( Origem,Destino,[],Caminho,Carreiras ).


profundidadeprimeiro( Destino,Destino,_,[],[] ) :- !.
profundidadeprimeiro( Origem,Destino,Historico,[ProxNodo|Caminho],[Carreira|Carreiras] )
    adjacente( Carreira,Origem,ProxNodo ),
    \+ memberchk(Origem/ProxNodo/Carreira,Historico ),
    profundidadeprimeiro( ProxNodo,Destino,[Origem/ProxNodo/Carreira|Historico],
    Caminho,Carreiras ).
```

Ainda de referir que ainda fiz uma pesquisa em profundidade com custos associados e ainda outra com um limite de profundidade para evitar que o programa entre em ciclos, mas isso pode ser visto no código que aconpanha o presente relatório.

• Pesquisa em largura

Já na pesquisa em largura (**Breadth-First Search**) a implementação já foi bastante diferente. Apesar de na secção Preliminares 2 já se ter referido a pesquisa em largura, brevemente o que se pode dizer é: a partir de uma origem e um destino determinar novamente o caminho e as carreiras por onde passou nesse trajeto, só que agora o que acontece é que além de ser necessário um histórico que guarda os nodos por onde já se passou, é necessário ainda uma orla, ou seja um conjunto de nodos que faltam visitar e são adjacentes ao nodos raiz que já passaram e já se encontram no histórico. Deste modo, o que eu fiz foi seguir as recomendações do professor nas aulas práticas, apesar de também ter tentado uma outra versão feita por mim, apresento de seguida a versão da abordagem que tivemos nas aulas práticas para este tipo de problema.

```
percurso_pl( Origem,Destino,Caminho,Carreiras ) :-
    larguraprimeiro( Destino,[(Origem/0,[],[])|Xs]-Xs,[],Caminho,Carreiras ).


larguraprimeiro( Destino,[(Nodo/Carr,Cams,Carrs)|_]-_,_,Result,ResultCarr ) :-
```

```
    Destino == Nodo,
    inverso( [Destino|Cams],Result ),
    inverso( Carrs,ResultCarr ).

larguraprimeiro( Destino,[(Nodo/Carr,_,_)|Xs]-Ys,Historico,Result,ResultCarr ) :-
    pertence( Nodo/Carr,Historico ),!,
    larguraprimeiro( Destino,Xs-Ys,Historico,Result,ResultCarr ).

larguraprimeiro( Destino,[(Nodo/Carr,Cams,Carrs)|Xs]-Ys,Historico,Result,ResultCarr ) :-
    findall( ProxNodo/Nodo/Carreira,adjacente(Carreira,Nodo,ProxNodo),Lista ),
    atualizar( Lista,Cams,Carrs,[Nodo/Carr|Historico],Ys-Zs ),
    larguraprimeiro( Destino,Xs-Zs,[Nodo/Carr|Historico],Result,ResultCarr ).


atualizar( [],_,_,_,X-X ).

atualizar( [ProxNodo/Nodo/Carreira|Resto],Cams,Carrs,Historico,Xs-Ys ) :-
    pertence( ProxNodo/Carreira,Historico ),!,
    atualizar( Resto,Cams,Carrs,Historico,Xs-Ys ).

atualizar( [ProxNodo/Nodo/Carr|Resto],Cams,Carrs,Historico,[(ProxNodo/Carr,[Nodo|Cams],
           [Carr|Carrs])|Xs]-Ys ) :-
    atualizar( Resto,Cams,Carrs,Historico,Xs-Ys ).
```

De igual modo, também existe uma versão para o percurso mas a mostrar o custo. Esse custo é medido em quilómetros, com uma aproximação pela distância euclidiana. Também se podia fazer o custo por tempo, e eu tenho uma implementação com tempo, mas preferi usar a distância em quilómetros porque achei que o custo deveria ser medido em quilómetros percorridos.

Deste modo, concluíu-se esta secção da Pesquisa Não Informada, pelo que apesar de não estarem aqui os algoritmos todos de pesquisa Não Informada, os que estão são apenas os que foram lecionados nas aulas práticas, pelo que penso que era este o objetivo.

3.3.2 Pesquisas Informadas

Nesta secção das pesquisas informadas, serão mostradas as duas técnicas de pesquisa abordadas nas aulas que são: a Pesquisa Gulosa(**Greedy Search**) e a **Pesquisa A***. Este tipo de pesquisa, como já referido anteriormente, são caracterizadas pela presença de heurísticas que servem para estimar custos de transporte para um dado destino, por exemplo. A presença de uma boa heurística para estes métodos será importante para se obterem os melhores resultados e até, quem sabe, obter à primeira a solução ótima. De seguida, serão abordadas mais em detalhe todos estes algoritmos de pesquisa informada.

• Pesquisa Gulosa

A Pesquisa Gulosa caracteriza-se pelo uso das estimativas para tentar achar, da maneira mais rápida possível, a solução, sendo essa estimativa feita apenas para os nodos adjecentes a um determinado nodo, não tendo portanto em considereção o trajeto que foi percorrido até ao momento, sendo por isso suscetível a falsos começos. Há ainda o problema de poder conduzir a ciclos, pelo que é necessário guardar a informação dos nodos por onde passou.

De seguida mostra-se o algoritmo usado para a pesquisa gulosa com base na heurística da distância euclidiana (*percurso_gulosabyDist*), contudo ainda tenho outra versão com outra heurística que basicamente baseia-se no menor número de nodos em média a percorrer até chegar ao destino.

```
percurso_gulosabyDist( Origem,Destino,Caminho,Carreiras,Custo ) :-
        estimabyDist( Origem,Destino,Estima ),
        gulosabyDist( Destino,[[Origem]/[]/0/Estima],Caminho1/Carreira1/Custo1/_ ),
        inverso( Caminho1,Caminho ),
        inverso( Carreira1,Carreiras ),
        converteKms( Custo1,Custo ),
        getParagensbyIds( Caminho,R ),
        escrever( R ), nl,
        write("Carreiras usadas no trajeto = "), nl,
        escrever( Carreiras ), nl,
        write("Total Custo do Caminho em Kms = "),
        write(Custo), nl.

gulosabyDist( Destino,Caminhos,MelhorCaminho ) :-
        obtem_melhor_caminho_Gulosa( Caminhos,MelhorCaminho ),
        MelhorCaminho = [Destino|_]/_/_/_.

gulosabyDist( Destino,Caminhos,Caminho ) :-
        obtem_melhor_caminho_Gulosa( Caminhos,MelhorCaminho ),
        seleciona( MelhorCaminho,Caminhos,OutrosCaminhos ),
        expande_CaminhosbyDist( Destino,MelhorCaminho,ExpCaminhos ),
        append( OutrosCaminhos,ExpCaminhos,NovoCaminhos ),
        gulosabyDist( Destino,NovoCaminhos,Caminho ).


obtem_melhor_caminho_Gulosa( [Caminho],Caminho ) :- !.

obtem_melhor_caminho_Gulosa( [Caminho1/Carreira1/Custo1/Est1,_/_/Custo2/Est2|Caminhos],
                             MelhorCaminho ) :-
        Est1 =< Est2, !,
        obtem_melhor_caminho_Gulosa( [Caminho1/Carreira1/Custo1/Est1|Caminhos],
                             MelhorCaminho ).

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


expande_CaminhosbyDist( Destino,Caminho,ExpCaminhos ) :-
        findall( NovoCaminho,adjacente_InfobyDist( Destino,Caminho,NovoCaminho ),
                    ExpCaminhos ).


adjacente_InfobyDist( Destino,[Nodo|Caminho]/Carrs/Custo/_,[ProxNodo,Nodo|Caminho]/
                      [Car|Carrs]/NovoCusto/Estimativa ) :-
        adjacente_c( Car,Nodo,ProxNodo,Custo1 ),
        nao( pertence( ProxNodo,Caminho ) ),
```

```
        NovoCusto is Custo + Custo1,
        estimabyDist( ProxNodo,Destino,Estimativa ).
```

Devido a este tipo de pesquisa não ter em consideração o custo do caminho já efetuado até ao momento, isso pode trazer problemas, como já referidos anteriormente. Assim, a pesquisa A* vem como combater este problema.

• **Pesquisa A***

Na pesquisa A* o funcionamento é muito semelhante ao da pesquisa gulosa, contudo a única diferença é que são tidos em conta os custos do percurso feito até ao momento mais a estimativa de se chegar até ao destino. Deste modo, a chance de haver falsos positivos é pouca, uma vez que se a heurística usada for boa, o que me parece no nosso caso das paragens quando se usa a distância euclidiana, as chances de se obter à primeira a solução ótima do problema são muito grandes.

De seguida, tal como tem vindo a acontecer, é mostrado a resolução do algoritmo com as estimativas da distância, pelo que tal como na pesquisa gulosa, também foi feito um outro algoritmo usando o número de paragens, mas que não é mostrado neste documento. De notar ainda que alguns dos predicados chamados estão em cima quando se mostrou o algoritmo da pesquisa gulosa, porque são os mesmos.

```
percurso_aestrelabyDist( Origem,Destino,Caminho,Carreiras,Custo ) :-
        estimabyDist( Origem,Destino,Estima ),
        aestrelabyDist( Destino,[[Origem]/[]/0/Estima],Caminho1/Carreira1/Custo1/_ ),
        inverso( Caminho1,Caminho ),
        converteKms( Custo1,Custo ),
        inverso( Carreira1,Carreiras ),
        getParagensbyIds( Caminho,R ),
        escrever( R ), nl,
        write("Carreiras usadas no trajeto = "), nl,
        escrever( Carreiras ), nl,
        write("Total Custo do Caminho em Kms = "),
        write(Custo), nl.


aestrelabyDist( Destino,Caminhos,Caminho ) :-
        obtem_melhor_caminho_Estrela( Caminhos,Caminho ),
        Caminho = [Destino|_]/_/_/_.

aestrelabyDist( Destino,Caminhos,SolucaoCaminho ) :-
        obtem_melhor_caminho_Estrela( Caminhos,MelhorCaminho ),
        seleciona( MelhorCaminho,Caminhos,OutrosCaminhos ),
        expande_CaminhosbyDist( Destino,MelhorCaminho,ExpCaminhos ),
        append( OutrosCaminhos,ExpCaminhos,NovoCaminhos ),
        aestrelabyDist( Destino,NovoCaminhos,SolucaoCaminho ).

obtem_melhor_caminho_Estrela( [Caminho],Caminho ) :- !.

obtem_melhor_caminho_Estrela( [Caminho1/Carreira1/Custo1/Est1,_/_/Custo2/Est2|Caminhos],
                              MelhorCaminho ) :-
```

```
        Custo1 + Est1 =< Custo2 + Est2, !,
        obtem_melhor_caminho_Estrela( [Caminho1/Carreira1/Custo1/Est1|Caminhos],
                            MelhorCaminho ).

obtem_melhor_caminho_Estrela( [_|Caminhos],MelhorCaminho ) :-
        obtem_melhor_caminho_Estrela( Caminhos,
                                    MelhorCaminho ).
```

Deste modo, a pesquisa A* revela-se a mais eficiente de todas as técnicas abordadas neste documento devido a nos dar mais depressa a solução ótima, e mesmo que não seja a ótima, está muito perto dela.

Desta forma, termina-se esta secção das pesquisas necessárias para a resolução de problemas de pesquisa, pelo que agora o que ficava a faltar era responder às *querys* (consultas) pedidas no enunciado do trabalho, sendo que ainda foram feitos alguns extra que não vão ser referidos neste documento, mas que poderão ser vistos no código que o acompanha. Mas antes disso, de seguida apresenta-se uma tabela comparativa das diferentes estratégias de Pesquisa usadas no presente trabalho.

3.3.3 Tabela comparativa das estratégias de Pesquisa

De seguida é mostrada uma tabela com as principais diferenças entre as diferentes estratégias de pesquisa utilizadas, tanto informadas como não informadas.

Propriedades	***Algoritmos de Pesquisa***			
	Pesquisa em Profundidade	Pesquisa em Largura	Pesquisa Gulosa	*Pesquisa A**
Completo	Não, falha em espaços de profundidade infinita, com repetições (loops)	Sim, se o fator de ramificação for finito	Não, suscetível a falsos começos	Sim
Otimal	Não, devolve a primeira solução que encontra	Sim, se o custo de cada ida de uma paragem para outra for 1	Não encontra sempre a ótima	Sim, se a heurística for boa
Complexidade (Tempo)	O(b^m), mau se profundidade máx >profundidade da melhor solução	O(b^d) é exponencial na profundidade melhor solução	O(b^m), mas se a heurística for boa, diminui	O(b^m), mas se a heurística for boa, diminui
Complexidade (Espaço)	O(bm), espaço linear, o que não é mau	Guarda cada nó em memória O(b^d)	O(b^m), guarda todos os nodos na memória	O(b^m), guarda todos os nodos na memória

Assim, nesta tabela são bem visíveis as principais diferenças entre as várias estratégias de pesquisa usadas, onde se repara que a melhor estratégia para encontrar a melhor solução é a Pesquisa A*, apesar de depender muito da heurística. Ainda de notar que todas as estratégias tem um tempo exponencial que depende no número de dados, pelo que no caso das Paragens do concelho de Oeiras, realmente o volume de dados é elevado, pelo que por vezes se demora ainda um bocado até se encontrar a solução ótima.

3.4 Consultas Pretendidas no Enunciado

Quanto às consultas pretendidas por parte da equipa docente, todas elas foram feitas, sendo que algumas até aproveitam o facto de se terem desenvolvido as várias técnicas de pesquisa. Contudo, muitas das implementações (a maior parte) não usa diretamente os algoritmos de pesquisa que foram descritos anteriormente, porque é ineficiente fazer um *findall* de todas as soluções primeiro para depois se escolherem apenas as que interessam. Assim, os algoritmos foram adaptados para que as soluções sejam descobridas mais cedo e logo que se encontra a solução que obdece a todas as restrições, mediante aquilo que se pretendia.
Deste modo, as consultas nos dados dos *datasets* que eram pretendidas são:

1. Calcular um trajeto entre duas paragens;
2. Selecionar apenas algumas das operadoras de transporte para um determinado percurso;
3. Excluir um ou mais operadores de transporte para o percurso;
4. Identificar quais as paragens com o maior número de carreiras num determinado percurso;
5. Escolher o menor percurso (usando critério menor número de paragens);
6. Escolher o percurso mais rápido (usando critério da distância);
7. Escolher o percurso que passe apenas por abrigos com publicidade;
8. Escolher o percurso que passe apenas por paragens abrigadas;
9. Escolher um ou mais pontos intermédios por onde o percurso deverá passar.

Assim, é possível resolver todas estas *querys* que eram pretendidas a partir dos vários predicados que foram definidos para o efeito. Contudo, esse código não vai ser mostrado todo aqui neste documento, pois senão ficava muito grande. Deste modo, na secção Anexos A vai poder ser visto alguns exemplos das querys, mas não todos para não ficar muito extenso o relatório. Mas, resumidamente, todas as consultas envolvem pesquisas, pelo que o importante é o algoritmo de pesquisa funcionar.

3.4.1 Extras

Relativamente aos extras, foram feitos alguns, como a partir de um conjunto de carreiras, devolve-nos o caminho que passa por todas essas carreiras obrigatoriamente, ou até estimativas por tempo, contudo não serão mostrados no relatório, pelo que podem ser vistos nas bases de Conhecimento *PROLOG*.

Capítulo 4

Conclusões e Sugestões

Com este trabalho prático Individual o que eu consegui aprender foram diferentes formas de pesquisa sobre dados e como lidar com um grande conjunto de dados complexos em *PROLOG*, pelo que penso que no geral este trabalho, que veio substituir um teste, foi bastante produtivo para desenvolver capacidades na programação em lógica.

Contudo, penso que os conjunto de dados fornecido era realmente bastante complexo, o que exigiu um grande tempo para lidar com eles e conseguir pô-los de tal forma que fossem possíveis usar no *PROLOG*. Mas depois de tudo isto feito, as pesquisas que foram sendo feitas foram dando bem para alguns casos, mas para outros demoravam muito tempo, talvez devido à grande profundidade dos dados existentes nas bases de Conhecimento das Paragens e das Adjacências.

Na realidade, as diferentes estratégias de Pesquisa poderiam ser melhoradas com a introdução de alguns critérios que possam diminuir esta complexidade e assim obter soluções em tempo de execução. Por isso, eu ainda tive o cuidado de desenvolver alguns critérios, mas penso que isso poderia ser melhorado no futuro.

Assim, a ideia com que acabo é que um trabalho é sempre feito sobre constante evolução, pelo que por vezes quando existe um prazo de entrega as coisas poderão não estar perfeitas, mas estão o máximo que uma pessoa pode dar. Desta forma, no futuro, todo este trabalho poderia ser melhorado de tal forma a estar 100% funcional e poder ser usado até, quem sabe, no quotidiano das pessoas.

Referências

[Bra00] Ivan Bratko. *PROLOG: Programming for Artificial Intelligence.* Addison-Wesley Longman Publishing Co., Inc., 3rd Edition, 2000.

[dS20] Equipa Docente de SRCR. Documento de apoio individual. BlackBoard, Universidade do Minho, 5 2020. Sistema de Transportes Públicos de Oeiras.

[RN15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* 3rd Edition, 2015.

Apêndice A

Anexos

A.1 Código *JAVA*

De seguida são mostrados apenas os principais métodos de leitura e escrita, sendo que ainda existem outros que são usados mas não são mostrados, pelo que se encontram no código fonte.

A.1.1 Leitura do Ficheiro das Paragens

```
/**
 * Método usado para ler toda a informação contida no ficheiro Excel
 * de Paragens de Autocarros e criar os respetivos objetos paragens
 */
public void leituraFileParagens(){

    int numParagens = 0;
    String aux = "";
    List<String> lAux;
    try {
        // obter o apontador para o ficheiro
        FileInputStream arquivo = new FileInputStream(new File(
                PARAGENS_FILE));

        // Objeto com o ficheiro excel
        XSSFWorkbook workbook = new XSSFWorkbook(arquivo);

        // Folha do Excel com indice 0
        XSSFSheet sheetParagens = workbook.getSheetAt(0);

        /* Enquanto houver linhas para ler*/
        for (Row row : sheetParagens) {
            if(numParagens>0) {
                // Serve para iterar sobre as colunas da folha
                Paragem paragem = new Paragem();
                /* Enquanto houver colunas para ler*/
                for (Cell cell : row) {
                    switch (cell.getColumnIndex()) {
                        case 0:
```

```
    // Gid
    paragem.setGid(Integer.parseInt(cell.getStringCellValue()));
    break;
case 1:
    // Latitude
    paragem.setLatitude(Double.parseDouble(cell.getStringCellValu
    break;
case 2:
    // Longitude
    paragem.setLongitude(Double.parseDouble(cell.getStringCellVal
    break;
case 3:
    // Estado de Conservacao
    if((aux = cell.getStringCellValue()).equals("")) {
        // Tratamento dos Valores Desconhecidos
        paragem.setEstado("xptoSRCR" + this.numDesconhecidos);
        if((lAux = this.valoresDesconhecidosP.get("Estado"))==nul
            lAux = new ArrayList<>();
        lAux.add("xptoSRCR" + this.numDesconhecidos);
        this.valoresDesconhecidosP.remove("Estado");
        this.valoresDesconhecidosP.put("Estado",lAux);
        this.numDesconhecidos++;
    } else {
        paragem.setEstado(this.preparaString(aux));
    }
    break;
case 4:
    // Tipo de Abrigo
    if((aux = cell.getStringCellValue()).equals("")) {
        // Tratamento dos Valores Desconhecidos
        paragem.setTipo_Abrigo("xptoSRCR" + this.numDesconhecidos
        if((lAux = this.valoresDesconhecidosP.get("Tipo de Abrigo
            lAux = new ArrayList<>();
        lAux.add("xptoSRCR" + this.numDesconhecidos);
        this.valoresDesconhecidosP.remove("Tipo de Abrigo");
        this.valoresDesconhecidosP.put("Tipo de Abrigo",lAux);
        this.numDesconhecidos++;
    } else {
        paragem.setTipo_Abrigo(this.preparaString(aux));
    }
    break;
case 5:
    // Abrigo com Publicidade
    boolean pub = false;
    if(cell.getStringCellValue().equals("Yes")) pub = true;
    paragem.setPublicidade(pub);
    break;
case 6:
    // Operadora
    if((aux = cell.getStringCellValue()).equals("")) {
```

```
                // Tratamento dos Valores Desconhecidos
                paragem.setOperadora("xptoSRCR" + this.numDesconhecidos);
                if((lAux = this.valoresDesconhecidosP.get("Operadora"))==
                    lAux = new ArrayList<>();
                lAux.add("xptoSRCR" + this.numDesconhecidos);
                this.valoresDesconhecidosP.remove("Operadora");
                this.valoresDesconhecidosP.put("Operadora",lAux);
                this.numDesconhecidos++;
            } else {
                paragem.setOperadora(this.preparaString(aux));
            }
            break;
        case 7:
            // Carreira
            String value = cell.getStringCellValue();
            String[] values = value.split(",");
            List<Integer> carrs = new ArrayList<>();
            for (String s : values) {
                carrs.add(Integer.parseInt(s));
                this.carreiras.add(Integer.parseInt(s));
            }
            paragem.setCarreira(carrs);
            break;
        case 8:
            // Codigo da Rua
            if((aux = String.valueOf(cell.getNumericCellValue())).equals(
                // Tratamento dos Valores Desconhecidos
                paragem.setCodigo_rua("xptoSRCR" + this.numDesconhecidos)
                if((lAux = this.valoresDesconhecidosP.get("Codigo da Rua"
                    lAux = new ArrayList<>();
                lAux.add("xptoSRCR" + this.numDesconhecidos);
                this.valoresDesconhecidosP.remove("Codigo da Rua");
                this.valoresDesconhecidosP.put("Codigo da Rua",lAux);
                this.numDesconhecidos++;
            } else {
                paragem.setCodigo_rua(this.preparaString(aux));
            }
            break;
        case 9:
            // Nome da Rua
            if((aux = cell.getStringCellValue()).equals("")) {
                // Tratamento dos Valores Desconhecidos
                paragem.setNome_rua("xptoSRCR" + this.numDesconhecidos);
                if((lAux = this.valoresDesconhecidosP.get("Nome da Rua"))
                    lAux = new ArrayList<>();
                lAux.add("xptoSRCR" + this.numDesconhecidos);
                this.valoresDesconhecidosP.remove("Nome da Rua");
                this.valoresDesconhecidosP.put("Nome da Rua",lAux);
                this.numDesconhecidos++;
            } else {
```

```
                                paragem.setNome_rua(this.preparaString(aux));
                            }
                            break;
                        case 10:
                            // Freguesia
                            if((aux = cell.getStringCellValue()).equals("")) {
                                // Tratamento dos Valores Desconhecidos
                                paragem.setFreguesia("xptoSRCR" + this.numDesconhecidos);
                                if((lAux = this.valoresDesconhecidosP.get("Freguesia"))==
                                    lAux = new ArrayList<>();
                                lAux.add("xptoSRCR" + this.numDesconhecidos);
                                this.valoresDesconhecidosP.remove("Freguesia");
                                this.valoresDesconhecidosP.put("Freguesia",lAux);
                                this.numDesconhecidos++;
                            } else {
                                paragem.setFreguesia(this.preparaString(aux));
                            }
                            break;
                    }
                }
                this.paragens.add(paragem);
            }
            numParagens++;
        }
        arquivo.close();

        /*for(Paragem p : paragens)
            System.out.println(p.toString());*/

        // Nem todas as carreiras tem correspondencia no adjacencias!!
        /*for(Integer s : this.carreiras)
            System.out.println("Carreira : " + s);*/

        System.out.println("Número Total de Paragens: " + numParagens);
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Arquivo Excel não encontrado!");
    }
}
```

A.1.2 Leitura do Ficheiro das Adjacências

```
/**
* Método usado para ler toda a informação contida no ficheiro Excel de listas
* de Adjacencias de Paragens e criar os respetivos objetos carreiras
*/
public void leituraFileAdjacencias(){
    int numParagens = 0;
```

```
String aux = "";
List<String> lAux;
try {
    // obter o apontador para o ficheiro
    FileInputStream arquivo = new FileInputStream(new File(
            ADJACENCIAS_FILE));

    // Objeto com o ficheiro excel
    XSSFWorkbook workbook = new XSSFWorkbook(arquivo);

    XSSFSheet sheetParagens;
    /*Percorro todas as folhas do ficheiro*/
    for(int i = 0;i<NUM_FOLHAS_ADJACENCIAS; i++) {
        sheetParagens = workbook.getSheetAt(i);
        if (this.carreiras.contains(Integer.parseInt(sheetParagens.getSheetName()))) {
            Carreira carreira = new Carreira();
            carreira.setId(Integer.parseInt(sheetParagens.getSheetName()));
            /* Enquanto houver linhas para ler*/
            for (Row row : sheetParagens) {
                Paragem paragem = new Paragem();
                if (numParagens > 0) {
                    /* Enquanto houver colunas para ler*/
                    for (Cell cell : row) {
                        switch (cell.getColumnIndex()) {
                            case 0:
                                // Gid
                                paragem.setGid(Integer.parseInt(cell.getStringCellVal
                                break;
                            case 1:
                                // Latitude
                                paragem.setLatitude(Double.parseDouble(cell.getString
                                break;
                            case 2:
                                // Longitude
                                paragem.setLongitude(Double.parseDouble(cell.getStrin
                                break;
                            case 3:
                                // Estado de Conservacao
                                if((aux = cell.getStringCellValue()).equals("")) {
                                    // Tratamento dos Valores Desconhecidos
                                    paragem.setEstado("xptoSRCR" + this.numDesconheci
                                    if((lAux = this.valoresDesconhecidosA.get("Estado
                                        lAux = new ArrayList<>();
                                    lAux.add("xptoSRCR" + this.numDesconhecidos);
                                    this.valoresDesconhecidosP.remove("Estado");
                                    this.valoresDesconhecidosA.put("Estado",lAux);
                                    this.numDesconhecidos++;
                                } else {
                                    paragem.setEstado(this.preparaString(aux));
                                }
```

```
    break;
case 4:
    // Tipo de Abrigo
    if((aux = cell.getStringCellValue()).equals("")) {
        // Tratamento dos Valores Desconhecidos
        paragem.setTipo_Abrigo("xptoSRCR" + this.numDesco
        if((lAux = this.valoresDesconhecidosA.get("Tipo d
            lAux = new ArrayList<>();
        lAux.add("xptoSRCR" + this.numDesconhecidos);
        this.valoresDesconhecidosA.remove("Tipo de Abrigo
        this.valoresDesconhecidosA.put("Tipo de Abrigo",l
        this.numDesconhecidos++;
    } else {
        paragem.setTipo_Abrigo(this.preparaString(aux));
    }
    break;
case 5:
    // Abrigo com Publicidade
    boolean pub = false;
    if (cell.getStringCellValue().equals("Yes")) pub = tr
    paragem.setPublicidade(pub);
    break;
case 6:
    // Operadora
    if((aux = cell.getStringCellValue()).equals("")) {
        // Tratamento dos Valores Desconhecidos
        paragem.setOperadora("xptoSRCR" + this.numDesconh
        if((lAux = this.valoresDesconhecidosA.get("Operad
            lAux = new ArrayList<>();
        lAux.add("xptoSRCR" + this.numDesconhecidos);
        this.valoresDesconhecidosA.remove("Operadora");
        this.valoresDesconhecidosA.put("Operadora",lAux);
        this.numDesconhecidos++;
    } else {
        paragem.setOperadora(this.preparaString(aux));
    }
    break;
case 7:
    // Carreira
    String value = cell.getStringCellValue();
    String[] values = value.split(",");
    List<Integer> carreiras = new ArrayList<>();
    for (String s : values)
        carreiras.add(Integer.parseInt(s));
    paragem.setCarreira(carreiras);
    break;
case 8:
    // Codigo da Rua
    if((aux = String.valueOf(cell.getNumericCellValue()))
        // Tratamento dos Valores Desconhecidos
```

```
                                paragem.setCodigo_rua("xptoSRCR" + this.numDescon
                                if((lAux = this.valoresDesconhecidosA.get("Codigo
                                    lAux = new ArrayList<>();
                                lAux.add("xptoSRCR" + this.numDesconhecidos);
                                this.valoresDesconhecidosA.remove("Codigo da Rua"
                                this.valoresDesconhecidosA.put("Codigo da Rua",lA
                                this.numDesconhecidos++;
                            } else {
                                paragem.setCodigo_rua(this.preparaString(aux));
                            }
                            break;
                        case 9:
                            // Nome de Rua
                            if((aux = cell.getStringCellValue()).equals("")) {
                                // Tratamento dos Valores Desconhecidos
                                paragem.setNome_rua("xptoSRCR" + this.numDesconhe
                                if((lAux = this.valoresDesconhecidosA.get("Nome d
                                    lAux = new ArrayList<>();
                                lAux.add("xptoSRCR" + this.numDesconhecidos);
                                this.valoresDesconhecidosA.remove("Nome da Rua");
                                this.valoresDesconhecidosA.put("Nome da Rua",lAux
                                this.numDesconhecidos++;
                            } else {
                                paragem.setNome_rua(this.preparaString(aux));
                            }
                            break;
                        case 10:
                            // Freguesia
                            if((aux = cell.getStringCellValue()).equals("")) {
                                // Tratamento dos Valores Desconhecidos
                                paragem.setFreguesia("xptoSRCR" + this.numDesconh
                                if((lAux = this.valoresDesconhecidosA.get("Fregue
                                    lAux = new ArrayList<>();
                                lAux.add("xptoSRCR" + this.numDesconhecidos);
                                this.valoresDesconhecidosA.remove("Freguesia");
                                this.valoresDesconhecidosA.put("Freguesia",lAux);
                                this.numDesconhecidos++;
                            } else {
                                paragem.setFreguesia(this.preparaString(aux));
                            }
                            break;
                    }
                }
                carreira.addParagem(paragem);
            }
            numParagens++;
            //System.out.println("Paragens: " + numParagens);
        }
        this.adjacencias.add(carreira);
    }
```

```
            numParagens=0;
        }
        arquivo.close();

    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Arquivo Excel não encontrado!");
    }
}
```

A.1.3 Escrita do Ficheiro das Paragens

```
/**
 * Método que serve para escrever os dados das paragens no ficheiro da base
 * de conhecimento
 * Formato: paragem( Gid,Lat.Long,Estado,Tipo,Pub,Oper,Carr,Cod,Nome,Freg ).
 */
public void escreveParagens(){

    try {
        // Abertura do ficheiro para escrita
        PrintWriter pw = new PrintWriter(NOME_FICH_PAR);
        /* Enquanto houverem paragens para escrever */
        for(Paragem p:this.paragens){
            // Preparacao da linha a inserir no ficheiro
            StringBuilder sb = new StringBuilder();
            sb.append("paragem( ");
            sb.append(p.getGid()); sb.append(",");
            sb.append(p.getLatitude()); sb.append(",");
            sb.append(p.getLongitude()); sb.append(",");
            sb.append(p.getEstado()); sb.append(",");
            sb.append(p.getTipo_Abrigo()); sb.append(",");
            if(p.isPublicidade()) sb.append("yes,");
            else sb.append("no,");
            sb.append(p.getOperadora()); sb.append(",");
            sb.append(p.getCarreira()); sb.append(",");
            sb.append(p.getCodigo_rua()); sb.append(",");
            if(!p.getNome_rua().contains("xptoSRCR")){
                sb.append("'"); sb.append(p.getNome_rua()); sb.append("'");
            } else sb.append(p.getNome_rua());
            sb.append(",");
            if(!p.getFreguesia().contains("xptoSRCR")) {
                sb.append("'"); sb.append(p.getFreguesia()); sb.append("'");
            } else sb.append(p.getFreguesia());
            sb.append(" ).");
            // Escrita da linha no ficheiro
            pw.println(sb.toString());
        }
        if(!this.valoresDesconhecidosP.isEmpty()){
```

```
            pw.println("\n% Excecoes relativas aos valores Desconhecidos presentes nesta
            for(String tipo:this.valoresDesconhecidosP.keySet()){
                List<String> lista = this.valoresDesconhecidosP.get(tipo);
                for(String nomeDesc:lista){
                    switch (tipo){
                        case "Estado":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                        case "Tipo de Abrigo":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                        case "Operadora":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                        case "Codigo da Rua":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                        case "Nome da Rua":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                        case "Freguesia":
                            pw.println("excecao( paragem( Gid,Lat,Long,Estado,Tipo,Pub,Op
                            break;
                    }
                }
            }
        }
        pw.close();
    } catch(IOException e){
        System.out.println("Erro de escrita no ficheiro de Paragens!!!");
    }
}
```

A.1.4 Escrita do Ficheiro das Adjacências

```
/**
 * Método que serve para escrever os dados das adjacencias no ficheiro da base
 * de conhecimento
 * Formato : adjacencia( Carreira,ParagemA,ParagemB,Distancia ).
 */
public void escreveAdjacencias(){
    Paragem pAtual, pProx;
    try {
        // Abertura do ficheiro para escrita
        PrintWriter pw = new PrintWriter(NOME_FICH_ADJ);
        /* Enquanto houverem paragens para escrever */
        for(Carreira c:this.adjacencias){
            for(int i=0;i<c.getParagens().size()-1;i++) {
                pAtual = c.getParagens().get(i);
                pProx = c.getParagens().get(i+1);
```

```
                // Preparacao da linha a inserir no ficheiro
                StringBuilder sb = new StringBuilder();
                sb.append("adjacencia( ");
                sb.append(c.getId());
                sb.append(",");
                sb.append(pAtual.getGid());
                sb.append(",");
                sb.append(pProx.getGid());
                sb.append(",");
                double dist = this.distanciaEntreParagens(pAtual.getLatitude(),pAtual.getL
                                                          pProx.getLatitude(),pProx.getLo
                sb.append(dist);
                sb.append(" ).");
                // Escrita da linha no ficheiro
                pw.println(sb.toString());
            }
        }
        pw.close();
    } catch(IOException e){
        System.out.println("Erro de escrita no ficheiro de Adjacencias!!!");
    }
}
```

A.2 Consultas *PROLOG*

De seguida são mostradas algumas das consultas nos dados que eram pretendidas por parte da equipa docente, não estão todas simplesmente para o ficheiro não ficar muito grande.

A.2.1 Excluir uma ou mais operadoras de transporte para o percurso

```
%------------------------------- - - - - - - - - - -  -  -  -  -   -
% >Excluir uma ou mais operadoras de transporte para o percurso
%------------------------------- - - - - - - - - - -  -  -  -  -   -

excluir_Operadoras( Origem,Destino,Operadoras,Caminho,Carreiras,Custo ) :-
            nao( verificaOperadoras_Cam( [Origem,Destino],Operadoras,P ) ),
            larguraprimeiro_noOperadoras( Destino,Operadoras,[(Origem/0/100000, [], [], 0
                        converteKms( Custo1,Custo ).

larguraprimeiro_noOperadoras( Destino,_,[(Nodo/Carr/C, Cams, Carrs, Custo)|_]-_,_,Result,
            Destino == Nodo,
            !,inverso( [Destino|Cams],Result ),
            inverso( Carrs,ResultCarr ).

larguraprimeiro_noOperadoras( Destino,Operadoras,[(Nodo/Carr/C, _,_,_)|Xs]-Ys,Historico,R
            pertence( Nodo/Carr/C,Historico ),!,
            larguraprimeiro_noOperadoras( Destino,Operadoras,Xs-Ys,Historico,Result,Resul

larguraprimeiro_noOperadoras( Destino,Operadoras,[(Nodo/Car/C2, Cams, Carrs,C)|Xs]-Ys,His
            findall( ProxNodo/Carr/C1,
```

```
            ( adjacente_c(Carr,Nodo,ProxNodo,C1),nao( verificaOperadoras_Cam( [Nodo,ProxN
            Lista ),
            atualizar_c( Nodo,C,Lista,Cams,Carrs,[Nodo/Car/C2|Historico],Ys-Zs ),
            larguraprimeiro_noOperadoras( Destino,Operadoras,Xs-Zs,[Nodo/Car/C2|Historico
```

A.2.2 Excluir uma ou mais operadoras de transporte para o percurso

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% >Escolher o percurso que passe apenas por abrigos com publicidade
%-------------------------------- - - - - - - - - - -  -  -  -  -   -

paragens_comAbrigoPublicidade( Origem,Destino,[Origem|Caminho],Carreiras ) :-
                        percursoPubAbrigo( Origem,Destino,[],Caminho,Carreiras ).

percursoPubAbrigo( Destino,Destino,_,[],[] ) :- existePublicidade(Destino), !.
percursoPubAbrigo( Origem,Destino,Historico,[ProxNodo|Caminho],[Carreira|Carreiras] ) :-
                        adjacente( Carreira,Origem,ProxNodo ),
                        existePublicidade( ProxNodo ),
                        nao( pertence( Origem/ProxNodo/Carreira,Historico ) ),
                        percursoPubAbrigo( ProxNodo,Destino,
                        [Origem/ProxNodo/Carreira|Historico],Caminho,Carreiras ).
```

A.2.3 Escolher um ou mais pontos intermédios por onde o percurso deverá passar

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% >Escolher um ou mais pontos intermedios por onde o percurso devera passar
%-------------------------------- - - - - - - - - - -  -  -  -  -   -

percurso_Intermedios( Origem,Destino,NodosIntermedios,[Origem|Caminho],Carreiras ) :-
                        ( ( pertence( Origem,NodosIntermedios ),
                        removeElem( Origem,NodosIntermedios,Resultado ),
                        percursoNodosInt( Origem,Destino,Resultado,[],Caminho,Carreiras )
                        percursoNodosInt( Origem,Destino,NodosIntermedios,[],Caminho,Carr


percursoNodosInt( Destino,Destino,[],_,[],[] ) :- !.

percursoNodosInt( Destino,Destino,_,_,[],[] ) :- !,fail.

percursoNodosInt( Origem,Destino,NodosIntermedios,Historico,[ProxNodo|Caminho],[Carreira|
                        adjacente( Carreira,Origem,ProxNodo ),
                        nao( pertence( Origem/ProxNodo/Carreira,Historico ) ),
                        ((pertence( ProxNodo,NodosIntermedios),
                        removeElem( ProxNodo,NodosIntermedios,NodosIntermedios1 ),
                        percursoNodosInt( ProxNodo,Destino,NodosIntermedios1,[Origem/Prox
                        (nao( pertence( ProxNodo,NodosIntermedios) ),
                        percursoNodosInt( ProxNodo,Destino,NodosIntermedios,[Origem/ProxN
```