UNIVERSIDADE DO MINHO

Trabalho Prático Individual

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre/2019-2020)

Rui Oliveira A83610

Braga
5 de Junho de 2020

Resumo

O presente relatório visa a documentação do trabalho individual proposto na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, que tem como objetivo a a utilização da Programação em Lógica, usando a linguagem de programação PROLOG, no âmbito de métodos de Resolução de Problemas e no desenvolvimento de algoritmos de pesquisa.

Ao longo deste documento, serão apresentadas as funcionalidades do sistema desenvolvido, bem como as estratégias utilizadas para execução das mesmas.

Conteúdo

Lista de Figuras

1 Introdução

Integrado na cadeira de Sistemas de Representação de Conhecimento e Raciocínios, foi proposto um trabalho prático de componente individual que tem como caso de estudo os dados do sistema de transportes do concelho de Oeiras.

Para a realização deste projeto, dividi o mesmo em duas fases. Primeiramente, foi necessário o desenvolvimento de um sistema, que permitisse importar os dados relativos às paragens de autocarros e representar os mesmos numa base de conhecimento. Posteriormente, foi desenvolvido um sistema de recomendação de transporte público para o caso de estudo.

Além do referido, foi proposto que a elaboração do caso prático permitisse:

1. Calcular um trajeto entre dois pontos;

2. Selecionar apenas algumas das operadoras de transporte para um determinado percurso;

3. Excluir um ou mais operadores de transporte para o percurso sistema;

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.

2 Preliminares

Para este trabalho, utilizei duas linguagens de programação, o *Java* para importação e processamento dos dados do caso de estudo e o *PROLOG* para o sistema de recomendação de transporte. Portanto, de modo a compreender o funcionamento das mesmas, é necessário compreender conceitos e significados das mesmas, no entanto, vou me focar naqueles relacionados com PROLOG, visto que foi a predominante ao longo da UC.

Primeiramente, para o sistema de recomendação de transporte é necessário entender que o mesmo se trata de um problema de procura. Portanto, é preciso compreender formulação de problemas, ou seja, procurar encontrar uma sequência de ações com início num determinado estado e que leva a um outro estado desejável. Para tal, é necessário formular e responder às seguintes perguntas:

- Quais as ações possíveis?
- Quais os estados possíveis? Como representá-los?
- Como avaliar os estados?

Além da formulação do problema, há a necessidade de haver uma familiarização com a forma como a pesquisa é feita para a resolução do mesmo. Através do espaço de estados do problema, é possível formar uma árvore de pesquisa que auxilie a encontrar a solução, sendo que o estado inicial forma a raíz da árvore e os ramos de cada nó são as ações possíveis a partir do nó(estado) para as folhas(próximos estados). Posto isto, existem várias estratégias para proceder à pesquisa de soluções, e os critérios para definir as mesmas são os seguintes:

- Completude: Está garantido que encontra solução?
- Otimalidade: Encontra a melhor solução?
- Complexidade no tempo: Quanto tempo demora encontrar a solução?
- Complexidade no espaço: Quanta memória necessita para fazer a pesquisa?

O tempo e a complexidade do espaço são medidas em termos de:

- b ,máximo fator de ramificação(o nº máximo de sucessores de um nó) da árvore de pesquisa.
- d, profundidade da melhor solução.
- m, máxima profundiade do espaço de estados.

De modo a compreender melhor a pesquisa de soluções, além dos critérios de avaliação, conhecer também os tipos de estratégias, ou seja, pesquisa não informada e informada. Neste campo, é necessário enteder a distinção entre as duas, sendo que a primeira usa apenas as informações disponíveis na definição do problema e a segunda recebe dicas sobre a adequação de diferentes estados.

Neste projeto, foram utilizadas diferentes estratégia de pesquisa não informada e informada, de modo a perceber melhor cada uma. De seguida, apresento aqueles que elaborei e analisei:

- Primeiro em largura(ou Breadth First Search)

  A estratégia deste algoritmo de pesquisa não informada é expandir primeiro todos os nós de menor profundidade.

- Primeiro em prifundidade(ou Depth First Search)

  Este, por oposição ao primeiro, tem como estratégia expandir sempre um dos nós mais fundos da árvore, sendo também de pesquisa não informada

- Algoritmo A* algoritmo, a estratégia passa por evitar expandir caminhos que são caros, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até à solução, ou seja, $f(n) = g(n) + h(n)$.

Após formular o problema e a pesquisa da solução, a fase final é a execução.

3 Descrição do Trabalho e Análise de Resultados

Nesta secção, serão descritas as características e funcionamento dos sistemas de importação de dados e de recomendação de tranporte público para o caso de estudo. Todas as componentes serão auxiliadas por valores experimentais de modo a comprovar a veracidade dos valores obtidos.

Para a realização do projeto, foram fornecidos dois ficheiros excel com os dados a tratar: paragem_autocarros_oeiras_processado e lista_adjacencias_paragens, sendo estes analisados individualmente, mas seguindo o mesmo raciocínio. Acabei por optar pelo segundo ficheiro para importar os dados, pelo facto de se tratar da lista de adjacência das paragens de autocarro, já organizada por carreiras e de forma sequencial.

Relativamante à construção de algumas extensões de predicados no sistema de recomendação de transporte, foi necessária a utilização dos seguintes predicados auxiliares:

- insercao/1 que, caso corra tudo bem, insere um termo na base de conhecimento;
- solucoes/3 que devolve as soluções em formato de lista de termos segundo uma determinada questão;
- solucoesSemRep/3 que devolve as soluções em formato de lista de termos segundo uma determinada questão, sem elementos repetidos;
- cabeca_lista/2 que devolve o primeiro elemento de uma lista;
- comprimento/2 que calcula o tamanho de uma lista;
- ultimo/2 que identifica o último elemento de uma lista.
- inverso/2 que inverte uma lista.
- escolhe/3 que retira um elemento de uma lista.
- tem_elementos/2 que verifica se os valores de uma lista estão todos presentes noutra.
- maiorlista/2 que calcula o maior elemento de uma lista de elementos.
- maior/3 que calcula o maior elemento de dois elementos.
- write_aux/1 que escreve no terminal os valores de uma lista, para auxiliar na demonstração dos resultados obtidos.

3.1 Descrição do Sistema e Importação dos Dados

Considere-se que o panorama será caracterizado por conhecimento dado na forma que se segue:

- paragem: Id, Gid, Latitude, Longitude, EstadoDeConservacao, TipoDeAbrigo, AbrigoComPublicidade, Operadora, Carreira, CodigoDeRua, NomeDaRua, Freguesia $\rightsquigarrow \{\mathbb{V}, \mathbb{F}, \mathbb{D}\}$.

Aos dados apresentados pelo ficheiro excel, decidi acrescentar um novo relativamente ao ID de cada paragem, de modo a ter algo único que distinguisse as paragens, uma vez que através do Gid, por exemplo, podemos ter várias paragens associadas.

Após formulado o conhecimento que pretendia obter com a importação dos dados, passei ao tratamento dos mesmos. De modo a importar os dados do ficheiro excel, usei a linguagem de programação *Java*, no entanto, foi necessário ainda um processamento prévio dos dados devido a erros de origem gramatical ou sintaxe: palavras com acentos desformatados, em que os acentos foram retirados; espaços em branco, que foram preenchidos com informação de acordo com as paragens adjacentes; sinais de pontuação, como ',', '.' e '-', substituídos por '_'. Além disso, utilizei a *Apache POI* como API, de modo a auxiliar o processo de importação.

Figura 1: Exemplos de erros que tiveram de ser corrigidos

Posteriormente, após processados e importados os dados para um ficheiro *processado_lista.txt*, cada linha do mesmo ficou com o seguinte formato:

```
paragem(Id,Gid,Latitude,Longitude,EstadoDeConservacao, TipoDeAbrigo,
    AbrigoComPublicidade,Operadora,Carreira,CodigoDeRua,NomeDaRua,Freguesia).
```

Posto isto, procedi à utilização do *PROLOG*. Perspetivando a importação dos dados para a base de conhecimento, foram implementadas os seguintes predicados:

- **readData**, que lê o ficheiro processado(linha a linha) com os dados das paragens e insere-os na base de conhecimento:

```
readData :-
    open('C:/Users/ruinu/Desktop/processado_lista.txt', read, Str),
    read_file(Str,Lines),
    close(Str),
    insere_paragens(Lines).

read_file(Stream,[]) :-
    at_end_of_stream(Stream).

read_file(Stream,[X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X),
    read_file(Stream,L).
```

- **insere_paragens/1**, que através da stream criada pelo *readData*(com o predicado *open*) insere as paragens na base de conhecimento:

```
insere_paragens([]).
insere_paragens([X|T]) :- insercao(X), insere_paragens(T).
```

Para colocar qualquer tipo de questão ao programa PROLOG, começamos por consultar o ficheiro relativo a este exercício (em Windows: abrir o SICTUS e correr *consult('(...)individual.pl').*). De seguida, utilizamos o predicado *readData.* de modo a que os dados das paragens sejam carregadas para a base de conhecimento e, consequentemente, possamos trabalhar com os mesmos para o sistema de recomendação de transporte.

3.2 Formulação do Sistema de Recomendação de Transporte

Nesta secção, tratarei de abordar como procedi para a formulação do problema do sistema de recomendação de transporte. Para tal, recorri a um grafo com o seguinte predicado:

```
g(grafo(L, A)) :- pontos(L), ids_pontos(I), lista_arestas(I,A).
```

Inicialmente, defini os estados do problema, representados pelos vértices do grafo, através da lista dos id's geográficos das paragens, com o predicado *pontos*:

```
pontos(L) :-
    solucoes((Ponto),
        paragem(Id, Ponto, X, Y, E, T, A, O, C, CR, N, F),L).
```

Após isso, defini as transições de estados/ações possíveis do problema, representadas pelas arestas do grafo, através da lista dos id's das paragens e da lista das carreiras. Para tal, criei o predicado *lista_arestas*:

```
lista_arestas([],[]).
lista_arestas([X|T], L) :-
    cabeca_lista(T, Y),
    Y > 0,
    id_geo(Y, YGEO),
    id_geo(X, XGEO),
    carreira_igual(X, Y, I),
    I > 0,
    distancia_pontos(XGEO, YGEO, D),
    lista_arestas(T, L1),
    append([aresta(XGEO, YGEO, I, D, 0)],L1,L);
    lista_arestas(T, L1),
    append([],L1,L).
```

Como as paragens estão organizadas por carreira e de forma sequencial, para obter as arestas do grafo, comparo de forma recursiva duas paragens seguidas e, caso tenham a mesma carreira, então há uma aresta que junta estes dois pontos, caso tenham carreiras diferentes, quer dizer que não há uma aresta entre esses pontos. Para cada aresta armazeno os dois pontos que a representam, a distância entre os mesmos e ainda a carreira e a operadora do transporte entre os dois.

Relativamente à distância entre cada paragem, optei por usar a distância euclidiana entre dois pontos, assumindo uma função tempo com base nessa distância, por cada 1 Km - 1min.

Por fim, de modo a saber se dois vértices têm uma aresta em comum, ou seja, se são adjacentes, criei o predicado *adjacente*:

```
adjacente(X,Y,C,D,O,grafo(N, L)) :- member(aresta(X,Y,C,D,O), L).
adjacente(X,Y,C,D,O,grafo(N, L)) :- member(aresta(Y,X,C,D,O), L).
```

Todos os predicados auxiliares utilizados na abordagem desta secção, serão explicitados na seguinte.

3.3 Predicados

Nesta secção será explicitada e comprovada a construção da extensão dos predicados utilizados para o sistema de recomendação de transporte.

3.3.1 paragem/12

O predicado *paragem: Id, Gid, Latitude, Longitude, EstadoDeConservacao, TipoDeAbrigo, AbrigoComPublicidade, Operadora, Carreira, CodigoDeRua, NomeDaRua, Freguesia* $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$, $\mathbb{D}$} tem como objetivo caracterizar uma paragem existente na base de conhecimento através do seu identificador, identificador geográfico, latitude, longitude, estado de conservação, tipo de abrigo, se o abrigo tem publicidade, operadora, carreira, código de rua, nome da rua e freguesia.

Com o objetivo de consultar o conhecimento adquirido sobre paragem/12, podemos recorrer ao predicado '*listing(paragem).*'.

3.3.2 distancia_pontos/3

O distancia_pontos: Paragem, Paragem, Resultado $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} identifica a distância entre duas paragens.

```
| ?- distancia_pontos(183,595,D).
D = 0.6168783773970339 ?
```

Figura 2: A distância entre a paragem com gid 183 e com gid 595

3.3.3 lista_paragens/1

O lista_paragens: Lista $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} calcula a lista das paragens que se encontram na base de conhecimento.

3.3.4 lista_carreiras/2

O lista_carreiras: Gid, Lista $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} calcula a lista das carreiras de determinada paragem, através do seu identificador geográfico.

```
| ?- lista_carreiras(261,L).
L = [1,2,10] ?
yes
```

Figura 3: Lista das carreiras da paragem com Gid

3.3.5 quant_carreiras_pontos/2

O quant_carreiras_pontos: Lista, Resultado $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} calcula a lista do número de carreiras que passam em determinadas paragens.

3.3.6 id_geo/2

O id_geo: Id, Gid $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} dado um determinado id, devolve o id geográfico respetivo.

3.3.7 id_paragem/2

O id_paragem: Gid, Paragem $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} dado um determinado id, devolve a paragem respetiva.

3.3.8 carreira_igual/3

O carreira_igual: Id, Id, Carreira $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} verifica se duas paragens têm carreiras iguais, caso tenham devolve a carreira respetiva.

```
| ?- id_paragem(1,R).
R = [paragem(1,183,-103678.36,-96590.26,bom,fechado_dos_lados,yes,vimeca,1,286.0,rua_aquilino_ribeiro,carnaxide_e_queijas)] ?
yes
| ?- id_paragem(3,R).
R = [paragem(3,595,-103725.69,-95975.2,bom,fechado_dos_lados,yes,vimeca,1,354.0,rua_manuel_teixeira_gomes,carnaxide_e_queijas)] ?
yes
| ?- carreira_igual(1,3,C).
C = 1 ?
yes
```

Figura 4: As paragens com id 1 e 3 têm ambos a carreira 1

3.3.9 tem_publicidade/2

O tem_publicidade: Paragem $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$ verifica se uma paragem tem abrigo com publicidade. De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- id_paragem(15,P).
P = [paragem(15,597,-104058.98,-95839.14,bom,fechado_dos_lados,yes,vimeca,1,1137.0,rua_tene
nte_general_zeferino_sequeira,carnaxide_e_queijas)] ?
yes
| ?- tem_publicidade(953).
yes
```

Figura 5: A Paragem com Gid 953 tem publicidade

3.3.10 tem_abrigo/2

O tem_abrigo: Paragem$\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$ verifica se uma paragem tem abrigo. De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- id_paragem(48,P).
P = [paragem(48,610,-104998.77,-95557.54,bom,sem_abrigo,no,vimeca,1,1196.0,rua_carlos_belo_
morais,carnaxide_e_queijas)] ?
yes
| ?- tem_abrigo(610).
no
```

Figura 6: A Paragem com Gid 610 não tem abrigo

3.3.11 pontos/1

O pontos: Lista $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$ calcula a lista dos gid's das paragens, que são utilizados como vértices do grafo.

3.3.12 ids_pontos/1

O ids_pontos: Lista $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$ calcula a lista dos id's das paragens, que são utilizados para auxiliar a definição das arestas do grafo.

3.4 Algoritmos de Pesquisa

Para a resolução do problema formulado anteriormente foi, como já referido, necessária a criação de algoritmos de pesquisa.

3.4.1 Depth First Search

Primeiramente, preocupei-me em criar os algoritmos de pesquisa não informada, começando por un algoritmo de pesquisa em profundido, sendo que defini então o predicado algoritmo_dfs: Grafo, Inicio, Fim, Visitados, Caminho $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$.

```
algoritmo_dfs(G, A, B, T, [C]) :-
    cabeca_lista(T, B),
    C is B.

algoritmo_dfs(G, A, B, Historico, [C|Caminho]) :-
    adjacente(A,Y,CA,D,O,G),
    C is A,
    nao(member(Y, Historico)),
    algoritmo_dfs(G, Y, B, [Y|Historico], Caminho).
```

Este algoritmo escolhe o primeiro elemento da lista dos estados não expandidos e adiciona extensões de caminho à frente da lista de estados não expandidos.

3.4.2 Breadth First Search

Para a pesquisa em largura foi necessário o seguinte predicdo algoritmo_bfs: Grafo, Inicio, Fim, Extendidos, Caminho $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$.

```
algoritmo_bfs(G, A, B, [[Paragem|Caminho]|_], [Paragem|Caminho]) :-
cabeca_lista([Paragem|Caminho], B).

algoritmo_bfs(G, A, B, [Caminho|Caminhos], Solucao) :-
    extende(G, Caminho, NovosCaminhos),
    concatenar(Caminhos, NovosCaminhos, Aux),
    algoritmo_bfs(G, A, B, Aux, Solucao).

extende(G, [Paragem|Caminho], NovosCaminhos) :-
    findall([NovaParagem, Paragem|Caminho],
        (adjacente(Paragem,NovaParagem,CA,D,O,G),
         nao(member(NovaParagem, [Paragem|Caminho]))),
         NovosCaminhos),!.

extende(G, Caminho,[]).
```

Este algoritmo escolhe o primeiro elemento da lista dos estados não expandidos e adiciona extensões de caminho ao final da lista de estados não expandidos.

3.4.3 A estrela(A*)

Após a pesquisa não informado, foi necessária elaboração dos algoritmos de pesquisa informando, sendo que comeceu por definir o seguinte predicado algoritmo_a_estrela: Grafo, Inicio, Fim, Caminhados, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$}.

```
algoritmo_a_estrela(G, A, B, Caminhos, Caminho) :-
    melhor(Caminhos, Caminho),
    Caminho = [Paragem|_]/_/_,
    Paragem == B.

algoritmo_a_estrela(G, A, B, Caminhos, Solucao) :-
    melhor(Caminhos, Melhor),
    escolhe(Melhor, Caminhos, Outros),
    extende_estrela(G, B, Melhor, CaminhosAux),
    append(Outros, CaminhosAux, NovosCaminhos),
    algoritmo_a_estrela(G, A, B, NovosCaminhos, Solucao).

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

melhor([Caminho1/Distancia1/Estimativa1,_/Distancia2/Estimativa2|Caminhos], MelhorCaminho) :-
    Distancia1 + Estimativa1 =< Distancia2 + Estimativa2, !,
    melhor([Caminho1/Distancia1/Estimativa1|Caminhos], MelhorCaminho).

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

extende_estrela(G, B, Caminho, CaminhosAux) :-
    findall(NovoCaminho, adjacente_Aux(G,    B,Caminho,NovoCaminho),
        CaminhosAux).

adjacente_Aux(G, B,[Paragem|Caminho]/Distancia/_, [ProxParagem,Paragem|Caminho]/NovaDistancia/
    adjacente(Paragem, ProxParagem,ProxDistancia,D,O,G),
    \+ member(ProxParagem, Caminho),
    NovaDistancia is Distancia + ProxDistancia,
    distancia_pontos(ProxParagem, B, Estimativa).
```

3.4.4 Tabela Comparativa

Algoritmo	Completo	Otimal	Tempo(Complexidade)	Espaço(Complexidade)
Primeiro em Largura (BFS)	Sim	Não	O(b^d)	O(b^d)
Primeiro em Profundidade (DFS)	Não	Não	O(b^m)	O(b*m)
A estrela(A*)	Sim	Sim	O(b^d)	O(b^d)

3.5 Queries

Chegando a esta fase do projeto, vou proceder à explicitação das estratégias utilizadas para a elaboração das queries, sendo que as mesmas serão auxiliadas por valores experimentais.

3.5.1 caminho/4

O caminho: Grafo, Inicio, Fim, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} é responsável por calcular o trajeto entre dois pontos. Escolhi o algoritmo depth first para esta query, uma vez que acho que é o ideal para quando o problema tem muitas soluções e usada muito pouca memória.

```
caminho(G,A,B,C) :-
    algoritmo_dfs(G,A,B,[A],C).
```

De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- g(G), caminho(G,183,107,C),write_aux(C).
183
791
595
182
499
593
181
180
594
185
89
107
G = grafo([183,791,595,182,499,593,181,180,594|...],[aresta(183,791,1,0.08763541293336934,_
A),aresta(791,595,1,0.6986929317661744,_B),aresta(595,182,1,0.42198634634310755,_C),aresta(
182,499,1,2.003334049129104,_D),aresta(499,593,1,0.24501549440800457,_E),aresta(593,181,1,1
.7345336738731814,_F),aresta(181,180,1,0.1272539885425926,_G),aresta(180,594,1,0.5111090131
273474,_H),aresta(...)|...]),
C = [183,791,595,182,499,593,181,180,594,185|...] ?
yes
```

Figura 7: Caminho da paragem com Gid 183 para a com 107

3.5.2 operadoras/5

O operadoras: Grafo, Inicio, Fim, Operadoras, Caminho $\rightsquigarrow \{\mathbb{V}, \mathbb{F}\}$ serve para selecionar(com 1 como o argumento em T) ou excluir(com 0 como argumento em T) algumas operadoras de transporte para um determinado percurso.

```
caminho_op(_, _, A, [A|C1], _, [A|C1]).
caminho_op(T, G, A, [Y|C1], OS, C) :-
        T == 1,
        adjacente(X,Y,CA,D,O,G),
        member(O, OS),
        \+ memberchk(X,[Y|C1]),
        caminho_op(T, G,A,[X,Y|C1],OS,C);
        T == 0,
        adjacente(X,Y,CA,D,O,G),
        \+ memberchk(O, OS),
        \+ memberchk(X,[Y|C1]),
        caminho_op(T, G,A,[X,Y|C1],OS,C).

operadoras(T, G, A, B, O, C) :- caminho_op(T, G, A, [B], O, C).
```

De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- g(G), operadoras(1,G,183,595,[vimeca],C).
G = grafo([183,791,595,182,499,593,181,180,594|...],[aresta(183,791,1,0.08763541293336934,v
imeca),aresta(791,595,1,0.6986929317661744,vimeca),aresta(595,182,1,0.42198634634310755,_A)
,aresta(182,499,1,2.003334049129104,_B),aresta(499,593,1,0.24501549440800457,_C),aresta(593
,181,1,1.7345336738731814,_D),aresta(181,180,1,0.1272539885425926,_E),aresta(180,594,1,0.51
11090131273474,_F),aresta(...)|...]),
C = [183,791,595] ?
yes
```

Figura 8: Caminho da paragem com Gid 183 para a com Gid 595 com apenas vimeca como operadora de transporte

3.5.3 paragens_mais_carreiras/6

O paragens_mais_carreiras: Grafo, Inicio, Fim, Caminho , Paragens, Maior $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} serve para identificar quais as paragens com o maior número de carreiras num determinado percurso.

```
par_mais_carreiras([],_,[]).
par_mais_carreiras([H|T],X,M) :-
        lista_carreiras(H, L),
        comprimento(L,R),
        R == X,
        par_mais_carreiras(T, X, M1),
        append([H],M1,M);
        par_mais_carreiras(T, X, M1),
        append([],M1,M).

paragens_mais_carreiras(G, A, B, C, M, X) :-
        caminho(G, A, B, C),
        quant_carreiras_pontos(C, L),
        maiorlista(L, X),
        par_mais_carreiras(C, X, M).
```

De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- g(G), paragens_mais_carreiras(G,183,595,C,M,X).
G = grafo([183,791,595,182,499,593,181,180,594|...],[aresta(183,791,1,0.08763541293336934,_
A),aresta(791,595,1,0.6986929317661744,_B),aresta(595,182,1,0.421986346343107 55,_C),aresta(
182,499,1,2.003334049129104,_D),aresta(499,593,1,0.24501549440800457,_E),aresta(593,181,1,1
.7345336738731814,_F),aresta(181,180,1,0.1272539885425926,_G),aresta(180,594,1,0.5111090131
273474,_H),aresta(...)|...]),
C = [183,791,595],
M = [595],
X = 7 ?
yes
| ?- lista_carreiras(595,L).
L = [7.0,13.0,1,2,10,12,15] ?
yes
```

Figura 9: Caminho da paragem com Gid 183 para a com Gid 595 com a identificação das paragens com o maior número de carreiras no percurso respetivo, neste caso a paragem 595 com 7 carreiras associadas

3.5.4 caminho_menos_paragens/4

O caminho_menos_paragens: Grafo, Inicio, Fim, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} trata de identificar o menor percurso, usando o critério menor número de paragens.

```
caminho_menos_paragens(G, A, B, C) :-
    algoritmo_bfs(G, A, B, [[A]],L),
    inverso(L, C).
```

Utilizei o breadth first search para a procura, uma vez que ao usar como estratégia primeiro a expansão dos nós de menor profundidade, obtém sempre o percurso com menos paragens para atingir a solução.

De seguida, apresento um exemplo para os resultados obtidos:

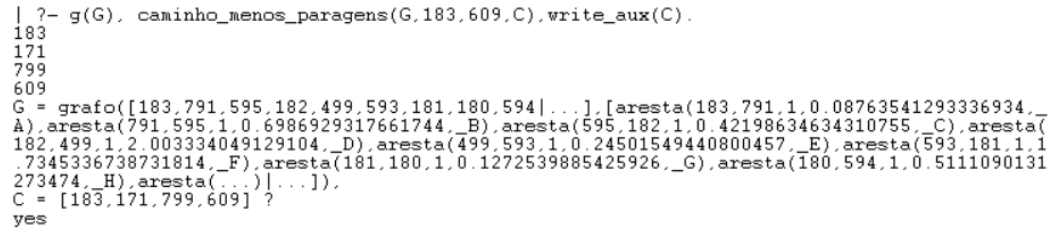

Figura 10: Caminho da paragem com Gid 183 para a com Gid 609 com menos paragens

3.5.5 caminho_mais_rapido/

O caminho_mais_rapido: Grafo, Inicio, Fim, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} trata de identificar o menor percurso, usando o critério menor número de paragens.

```
caminho_mais_rapido(G, A, B, Caminho/Custo) :-
    distancia_pontos(A, B, R),
    algoritmo_a_estrela(G, A, B, [[A]/0/R], Inv_Caminho/Custo/_),
    inverso(Inv_Caminho, Caminho).
```

Utilizei o best first search(A*) para a procura, uma vez que evita expandir caminhos que têm custos(neste caso, distâncias) maiores. De seguida, apresento um exemplo para os resultados obtidos:

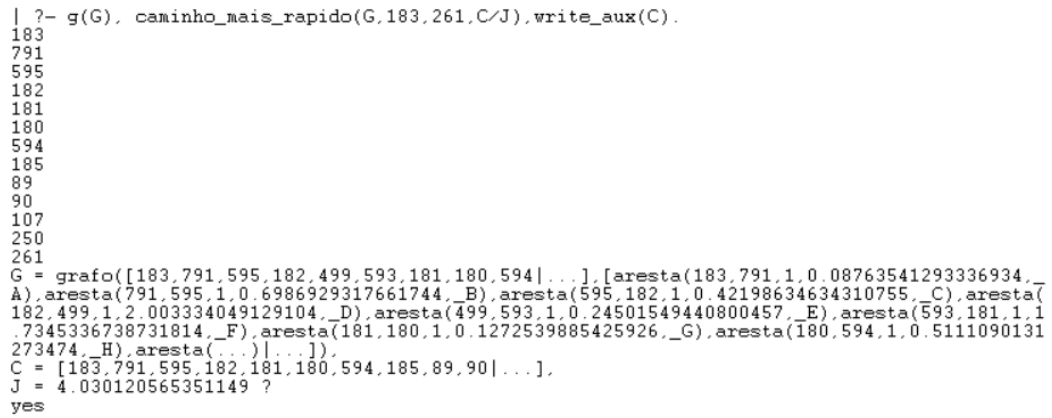

Figura 11: Caminho da paragem com Gid 183 para a com Gid 261 mais rápido

3.5.6 caminho_com_publicidade/4

O caminho_com_publicidade: Grafo, Inicio, Fim, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} é responsável por identificar um caminho que passe apenas por abrigos com publicidade.

```
com_publicidade(_,A,[A|C1],[A|C1]).
    com_publicidade(G,A,[Y|C1],C) :- adjacente(X,Y,CA,D,O,G),
                                tem_publicidade(X, yes),
                                \+ memberchk(X,[Y|C1]),
                                com_publicidade(G,A,[X,Y|C1],C).

caminho_com_publicidade(G, A, B, C) :- com_publicidade(G, A, [B], C).

caminho_com_publicidade_bf(G, A, B, CC) :-
    tem_publicidade(A),
    algoritmo_bfs_publicidade(G, A, B,[[A]], C),
    inverso(C, CC).
```

Defini o predicado de modo a que seja possível escolher o algoritmo a utilizar, depth first ou breadth first. De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- g(G), caminho_com_publicidade(G,183,181,C).
G = grafo([183,791,595,182,499,593,181,180,594|...],[aresta(183,791,1,0.08763541293336934,_
A),aresta(791,595,1,0.6986929317661744,_B),aresta(595,182,1,0.42198634634310755,_C),aresta(
182,499,1,2.003334049129104,_D),aresta(499,593,1,0.24501549440800457,_E),aresta(593,181,1,1
.7345336738731814,_F),aresta(181,180,1,0.1272539885425926,_G),aresta(180,594,1,0.5111090131
273474,_H),aresta(...)|...]),
C = [183,791,595,182,181] ?
yes
```

Figura 12: Caminho da paragem com Gid 183 para a com Gid 181 que passa apenas por abrigos com publicidade

3.5.7 caminho_com_abrigo/4

O caminho_com_abrigo: Grafo, Inicio, Fim, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} é responsável por identificar um caminho que passe apenas por paragens abrigadas.

```
com_abrigo(_,A,[A|C1],[A|C1]).
    com_abrigo(G,A,[Y|C1],C) :- adjacente(X,Y,CA,D,O,G),
                        tem_abrigo(X, sem_abrigo),
                        \+ memberchk(X,[Y|C1]),
                        com_abrigo(G,A,[X,Y|C1],C).

caminho_com_abrigo(G, A, B, C) :- com_abrigo(G, A, [B], C).

caminho_com_abrigo_bf(G, A, B, CC) :-
    tem_abrigo(A),
    algoritmo_bfs_abrigo(G, A, B,[[A]], C),
    inverso(C, CC).
```

Defini o predicado de modo a que seja possível escolher o algoritmo a utilizar, depth first ou breadth first. De seguida, apresento um exemplo para os resultados obtidos:

```
| ?- g(G), caminho_com_abrigo(G,183,180,C).
G = grafo([183,791,595,182,499,593,181,180,594|...],[aresta(183,791,1,0.08763541293336934,_
A),aresta(791,595,1,0.6986929317661744,_B),aresta(595,182,1,0.42198634634310755,_C),aresta(
182,499,1,2.003334049129104,_D),aresta(499,593,1,0.24501549440800457,_E),aresta(593,181,1,1
.7345336738731814,_F),aresta(181,180,1,0.1272539885425926,_G),aresta(180,594,1,0.5111090131
273474,_H),aresta(...)|...]),
C = [183,791,595,182,181,180] ?
yes
```

Figura 13: Caminho da paragem com Gid 183 para a com Gid 180 que passa apenas por paragens abrigadas

3.5.8 caminho_com_intermedios/5

O caminho_com_intermedios: Grafo, Inicio, Fim, Lista, Caminho $\rightsquigarrow$ {$\mathbb{V}$, $\mathbb{F}$} é responsável por identificar um caminho que passe por determinados pontos intermédios.

```
com_intermedios(_,A,[A|C1],[],[A|C1]).
com_intermedios(G,A,[Y|C1],C) :- adjacente(X,Y,CA,D,O,G),
                        member(X, C),
                        \+ memberchk(X,[Y|C1]),
                        com_intermedios(G,A,[X,Y|C1],C).

caminho_com_intermedios(G, A, B, L,  LL) :-
    algoritmo_dfs(G, A, B, [A],LL),
    tem_elementos(L, LL);
    algoritmo_bfs(G, A, B, [[A]],LL),
    tem_elementos(L, LL).
```

Este predicado tem a particularidade de usar os algoritmos depth first e breadth first de modo a identificar um caminho que passe por um ou mais pontos intermédios. De seguida, apresento um exemplo para os resultados obtidos:

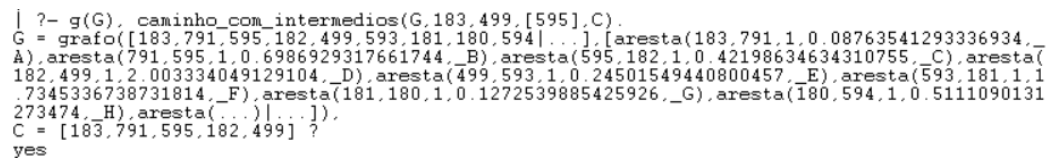

Figura 14: Caminho da paragem com Gid 183 para a com Gid 499 que passa pela paragem 595

4 Conclusões e Sugestões

Chegando à fase final deste relatório, considero que a realização deste trabalho prático individual foi bem sucedida, visto que acredito ter abordado e cumprido todos os requisitos iniciais e, além disso, procurei investir em novas funcionalidades, de modo a enriquecer o sistema de recomendação de transporte.

Numa fase posterior algo que poderia ser adicionado a este trabalho seria mais alguns algoritmos de pesquisa para poder comparar as suas propriedades com as dos já existentes.

Em suma, este exercício permitiu uma exploração e consolidação no que diz respeito a métodos de resolução de problemas e de procura dos métodos de pesquisa em programação lógica.