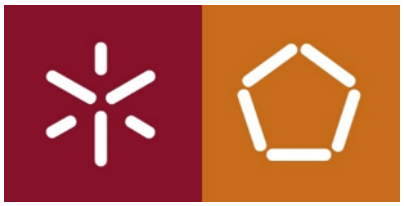

Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio
(3º ano de curso, 2º semestre)

Exercício 2

Relatório de Desenvolvimento

Mestrado Integrado em Engenharia Informática

Realizado pelo aluno:
Válter Ferreira Picas Carvalho - a84464

5 de Junho de 2020

Resumo

O segundo exercício proposto nesta UC consiste num sistema de transportes, constituído por paragens e carreiras, no concelho de Oeiras. Este sistema encontra-se disponível em repositório aberto no endereço Sistema de Transportes em Oeiras.

No contexto de Sistemas de Representação de Conhecimento e Raciocínio, o objetivo principal era desenvolver um programa utilizando o paradigma de programação em lógica para responder a uma série de interrogações sobre caminhos entre duas (ou mais) paragens. Para ser possível a sua realização, foi utilizado o mecanismo de interpretação **SWI-Prolog**, porém foram utilizadas bibliotecas de funções comuns a outros mecanismos como o **SICStus**.

Para estabelecer esta pesquisa de caminhos entre duas paragens foram utilizadas duas abordagens: uma com **pesquisa informada**, que estabelece a procura utilizando algum tipo de informação extra sobre o problema em todas as iterações, a outra com **pesquisa não-informada**, que apenas procura a solução sem saber nada extra sobre o problema em si, apenas para onde tem possibilidade de seguir em cada iteração.

Para comparar estes dois métodos de pesquisa será utilizada uma tabela de comparação e, no final, concluir acerca de qual o mais eficiente em casos mais gerais.

Conteúdo

Lista de Figuras

1 Introdução

O objetivo e principal propósito deste trabalho individual foi introduzir dois métodos de pesquisa totalmente diferentes: **pesquisa informada** e **pesquisa não-informada** para resolução de um problema típico de complexidade elevada, que é a procura em grafos.

O tema escolhido para este projeto foi a rede de transportes em Oeiras, fornecidos pela equipa docente e foi pedida a sua análise. Após uma perceção inicial do problema, era necessário desenvolver a sua consecutiva resolução simbólica para estes dois métodos de pesquisa.

Para ser possível a resolução do problema, foi utilizado o software **SWI-Prolog**, que é *open source* e possível integrar no *Visual Studio Code*, de modo a que seja mais simples compilar e trabalhar diretamente no código.

Visto que o propósito deste trabalho prático era a introdução a estes dois métodos de pesquisa, é necessário estabelecer uma relação de comparação entre os dois métodos, para concluir qual das duas será mais eficiente num cenário do dia-a-dia e, eventualmente, melhorar para casos mais complexos. Para comparar, será utilizada uma tabela com os tempos de execução em distintos cenários para estes dois métodos.

2 Preliminares

Para ser possível começar a resolução do problema, era necessário saber primeiro do que se tratava a **pesquisa informada** e a **pesquisa não-informada**, pelo que foi necessário um trabalho de pesquisa inicial para entender as diferenças entre ambas.

A **pesquisa informada** pretende, de alguma forma, buscar conhecimento específico ao problema para diminuir o tempo de procura e evitar iterações por caminhos que certamente nunca darão corretos, através duma heurística que permita transmitir em cada iteração este conhecimento obtido do problema.

A **pesquisa não informada**, por outro lado, apenas diferencia o estado inicial (no caso do problema em causa isso corresponde às paragens) e o estado final (que é tambem uma paragem), e é necessário uma heurística que vá procurando as diferentes possibilidades até o estado final ser atingido, sem utilizar conhecimento extra sobre o problema em causa.

Para ser possível o desenvolvimento da solução usando uma linguagem simbólica como a programação em lógica **Prolog**, os conhecimentos adquiridos nas aula práticas sobre grafos, métodos de pesquisa e estratégias de resolução de problemas revelaram-se imprescindíveis. Juntando este conhecimento ao motor de inferência **SWI-Prolog**, foi possível atingir os objetivos propostos.

Após esta fase inicial de pesquisa teórica, foi necessário proceder ao tratamento dos dados fornecidos e, para isso, foi utilizada a biblioteca **Pandas** disponível para *Python*, que era uma linguagem que não foi abordada no contexto universitário e, portanto, era necessário aprendê-la, assim como as funções disponíveis nessa biblioteca, necessárias para carregar ficheiros CSV.

Por fim, foi finalmente iniciado o trabalho de desenvolvimento da resolução com todos os dados tratados e convertidos no formato final pretendido.

3 Descrição do Trabalho e Análise de Resultados

Este trabalho foi realizado tendo em conta alguns pressupostos, visto que não se impunha limitações na interpretação do problema em causa. Uma vez que a pesquisa informada aproveita toda a informação importante e pertinente do problema, era necessário criar um conjunto de restrições que fizessem sentido num contexto real.

Portando, foi decidido resolver o problema num contexto mais "real":

1. O utilizador só muda de autocarro no final da carreira;
2. Se possível, é preferível chegar ao destino com uma única carreira (só paga potencialmente um passe para o transporte);
3. O utilizador sai em apenas uma paragem sempre que muda de carreira, ou seja num caminho que passe em 2 carreiras ele estará em 3 paragens no total (inicial, intermédia e final).

Para a pesquisa não informada, alguns destes pontos não é possível cumprir visto que não é possível ter um panorama geral do problema, mas na pesquisa informada o contrário é verdade, pelo que é muito favorecida nesse aspeto.

Durante os capítulos que se seguem serão discutidas as diferentes abordadas para as interrogações e a forma como foi lidada a leitura de dados dos ficheiros fornecidos, assim como as diferentes abordagens de algoritmos para o problema em questão.

3.1 Interpretação dos Ficheiros de Dados

Inicialmente, foi fornecido um ficheiro Excel que continha 39 folhas distintas, cada uma com as paragens relativas a uma determinada carreira. Foi necessário, em primeiro lugar, guardar cada uma das páginas singularmente em ficheiros CSV, terminando o processo com 39 ficheiros distintos com esta extensão, para ser mais simples implementar uma *Python Script* que faça a sua leitura.

Como mencionado anteriormente, para esta leitura foi utilizada a biblioteca **Pandas** e a sua diversa flexibilidade com *dataframes* permitiu de forma simples realizar este processo.

Para cada *dataframe* relativo a cada um dos 39 ficheiros, foram retiradas todas as linhas, que dizem respeito às paragens, e guardadas como predicados na base de conhecimento com o seguinte formato:

- paragem(ID da Carreira, ID da Paragem, GID, Latitude, Longitude, Estado de Conservação, Tipo de Abrigo, Abrigo com Publicidade, Operadora, Código de Rua, Nome da Rua, Freguesia).
- adjacente(GID 1, GID 2).

Neste formato, foi inserido um novo campo relativamente aos fornecidos no documento: o **ID da Paragem**. Este campo serve para identificar, numa carreira, a ordem sequencial das paragens, que será útil no algoritmo implementado, visto que o GID não é incremental nem tem uma ordem concretamente definida.

Para o caso do predicado *adjacente*, é feito para simular um arco num grafo, ou seja, existe ligação direta entre as paragens com GID 1 e com GID 2 como suas identificadoras.

Todo este código está disponível no Anexo para uma análise mais detalhada, assim como exemplos do resultado final da sua execução.

3.2 Algoritmo com Pesquisa Informada

O algoritmo base para a resolução do problema parte dos pressupostos enunciados no início deste capítulo e foca-se em otimizar para as situações descritas, utilizando pesquisa informada.

É tirado partido da geografia das paragens, pelo que se nota nos Anexos na secção visual do grafo, que as carreiras interligadas têm uma carreira intermediária que liga duas grandes redes de paragens e carreiras. Há dois pormenores muito importantes: nessas duas grandes redes, todas as paragens estão no máximo separadas por 1 carreira intermédia e que se não é possível fazer a conexão entre duas paragens, então estão em "redes"distintas, pelo que é preciso percorrer aquela carreira intermédia. É, também, **bi-direcional**.

O algoritmo tem, então, as seguintes vertentes numeradas por preferência, quando se pretende um caminho da paragem A para a paragem B:

1. Se a carreira(s) em que A está localizada for inacessível (não tiver nenhuma carreira externa que ligue nenhuma das paragens dessa carreira), o caminho é impossível. O

mesmo acontece para a paragem B;

```
% Retorna a lista de carreiras a que uma paragem pertence.
carreirasAdjacentes(Carreira1,Paragem,Lista):-
    findall(Carreira, (paragem(Carreira,_,Paragem, _, _, _, _, _, _, _,
        _, _), Carreira1 \= Carreira),
    Lista).

% Retorna a lista de carreiras adjacentes a uma determinada carreira.
carreirasAdjacentesCarreiraAux(_, [], []).

carreirasAdjacentesCarreiraAux(Carreira,[Paragem|Paragens],[Lista1|Lista
    ]):-
    carreirasAdjacentes(Carreira,Paragem,Lista1),
    carreirasAdjacentesCarreiraAux(Carreira,Paragens,Lista).

carreirasAdjacentesCarreira(Carreira, Lista):-
    findall(Paragem, (paragem(Carreira,_,Paragem, _, _, _, _, _, _, _, _
        , _)),Paragens),
    carreirasAdjacentesCarreiraAux(Carreira,Paragens,Lista).

% Predicado que testa se as paragens sao isoladas
caminho(_,Fim,_):-
    paragem(Carreira2,_ ,Fim, _, _, _, _, _, _, _, _, _),
    carreirasAdjacentesCarreira(Carreira2,Adjs2),
    flatten(Adjs2,Help),
    length(Help,0), %    uma carreira totalmente isolada
    !,fail.

caminho(Inicio,_,_):-
    paragem(Carreira1,_, Inicio, _, _, _, _, _, _, _, _, _),
    carreirasAdjacentesCarreira(Carreira1,Adjs),
    flatten(Adjs,Help),
    length(Help,0), %    uma carreira totalmente isolada
    !,fail.
```

2. Se a carreira em que A e B estiverem localizadas for idêntica, o caminho é estabelecido diretamente percorrendo as paragens intermédias dessa mesma carreira;

```
% Retorna a lista de elementos um Inicio e um Fim na mesma carreira.
criarCaminho(_,Fim,Fim,[Fim]):-!.

criarCaminho(Carreira,Inicio,Fim,[Inicio|Caminho]):-
    paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
    Id1 < IdAux,
    Id2 is Id1 + 1,
    paragem(Carreira,Id2, Adjacente, _, _, _, _, _, _, _, _, _),
    criarCaminho(Carreira, Adjacente, Fim, Caminho).

criarCaminho(Carreira,Inicio,Fim,[Inicio|Caminho]):-
```

```
    paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
    Id1 > IdAux,
    Id2 is Id1 - 1,
    paragem(Carreira,Id2, Adjacente, _, _, _, _, _, _, _, _, _),
    criarCaminho(Carreira, Adjacente, Fim, Caminho).

% Predicado que cria um caminho entre duas paragens, se a carreira for
    igual.
caminho(Inicio,Fim,Caminho):-
    paragem(Carreira,_, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,_ ,Fim, _, _, _, _, _, _, _, _, _),
    criarCaminho(Carreira,Inicio,Fim,Caminho),
    !.
```

3. Se A estiver numa carreira diferente de B mas essas carreiras possuírem uma paragem em comum, por exemplo a paragem C, com C diferente de A e de B, é criado um caminho de A para C e de C para B;

```
% Predicado que remova a cabeca de uma lista.
removehead([_|T], T).

% Predicado que cria um caminho entre duas paragens, se tiver um ponto
    de juncao comum
caminho(Inicio,Fim,Caminho):-
    paragem(Carreira1,_, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira2,_ ,Fim, _, _, _, _, _, _, _, _, _),
    carreirasAdjacentesCarreira(Carreira2,Adjs2),
    flatten(Adjs2,Help),
    member(Carreira1,Help), % procura se a carreira original esta nas
        atingiveis pela carreira que contem o no final
    paragem(Carreira1,_,Escolhido, _, _, _, _, _, _, _, _, _), % vou
        buscar esse comum
    paragem(Carreira2,_,Escolhido, _, _, _, _, _, _, _, _, _), % vou
        buscar esse comum
    criarCaminho(Carreira1, Inicio, Escolhido, Resultante1),
    criarCaminho(Carreira2, Escolhido, Fim, Resultante2x), % fica aqui
        com um elemento a mais, tira-se
    removehead(Resultante2x,Resultante2),
    append(Resultante1, Resultante2, Caminho).
```

4. Se A estiver numa carreira diferente de B mas essas carreiras possuírem uma carreira adjacente (todas as carreiras a que todas as paragens pertencem) que seja igual, então são selecionadas as paragens que permitem a ligação, isto é, é selecionada a paragem C na carreira que A pertence e a paragem D na carreira a que B pertence que ligam C e D nessa carreira comum e realiza-se o caminho de A para C, C para D e, no final, D para B.

```

```

```
% Retorna todos os elementos comuns a ambas as matrizes.
verificaMatrizAux([], _, []).

verificaMatrizAux(_, [], []).

verificaMatrizAux([L1|Resto],X, [L1|Res]):-
    member(L1, X),
    verificaMatrizAux(Resto,X,Res), !.

verificaMatrizAux([_|R],X, Res):-
    verificaMatrizAux(R,X,Res),
    !.

verificaMatriz(X,Y,Res):-
    flatten(X,F1),
    flatten(Y,F2),
    verificaMatrizAux(F1,F2,Res).

% Predicado que cria um caminho entre duas paragens, se tiverem alguma
    carreira em comum intermedia
caminho(Inicio,Fim,Caminho):-
    paragem(Carreira1,_, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira2,_ ,Fim, _, _, _, _, _, _, _, _, _),
    carreirasAdjacentesCarreira(Carreira1,Adjs1),
    carreirasAdjacentesCarreira(Carreira2,Adjs2),
    verificaMatriz(Adjs1,Adjs2,Comuns), % procura as carreiras
        adjacentes comuns
    member(Comum,Comuns), % retira um elemento comum
    paragem(Comum, _, Escolhido, _, _, _, _, _, _, _, _, _),
    paragem(Carreira1, _, Escolhido, _, _, _, _, _, _, _, _, _),
    paragem(Comum, _, Escolhido2, _, _, _, _, _, _, _, _, _),
    paragem(Carreira2, _, Escolhido2, _, _, _, _, _, _, _, _, _),
    criarCaminho(Carreira1,Inicio,Escolhido, Resultante1),
    criarCaminho(Comum,Escolhido,Escolhido2, Resultante2x),
    removehead(Resultante2x,Resultante2),
    criarCaminho(Carreira2,Escolhido2,Fim, Resultante3x),
    removehead(Resultante3x,Resultante3),
    append(Resultante1, Resultante2, CaminhoAux),
    append(CaminhoAux, Resultante3, Caminho).
```

5. Se nada funcionar das condições anteriores, é realizado o caminho de A para a paragem que começa a carreira intermédia (**GID=178**) e dessa paragem realiza-se o caminho para B, que pertence ao segundo grupo relativo de redes.

```
% Predicado que cria um caminho entre duas paragens n o antig veis
    pela informa  o acima, usando a paragem terminal
caminho(Inicio,Fim,Caminho):-
    caminho(Inicio,178,CAux),
    caminho(178,Fim,C2A),
    removehead(C2A,C2),
```

```
6       append(CAux,C2,Caminho).
```

3.2.1 Caminho entre Duas Paragens

Esta era a primeira interrogação no enunciado do trabalho prático e, para ser possível de a realizar, era necessário desenvolver o algoritmo que retorna pelo menos um caminho entre duas paragens. Não requer nada extra do algoritmo inicial, pelo que basta invocá-lo com os campos preenchidos corretamente e obter o resultado pretendido no final da execução.

```
9 ?- caminho(583,712,R).
R = [583, 584, 585, 576, 941, 581, 579, 969, 944, 577, 913, 967, 501, 505, 521, 310, 538,
518, 540, 10, 543, 516, 503, 542, 541, 419, 88, 869, 989, 379, 378, 294, 281, 283, 282, 28
5, 284, 828, 796, 795, 191, 797, 208, 785, 781, 756, 762, 778, 777, 773, 774, 776, 775, 77
9, 751, 780, 732, 152, 744, 715, 127, 125, 711, 125, 127, 715, 744, 152, 732, 733, 151, 74
3, 739, 158, 157, 84, 83, 690, 738, 708, 1015, 1016, 169, 789, 817, 692, 693, 178, 947, 79
2, 710, 807, 818, 823, 952, 954, 1002, 977, 986, 983, 354, 353, 863, 856, 857, 367, 333, 8
46, 845, 330, 364, 33, 32, 60, 61, 64, 63, 62, 58, 57, 59, 654, 655, 56, 491, 490, 458, 45
7, 335, 363, 344, 345, 343, 346, 342, 341, 85, 86, 347, 339, 352, 351, 323, 313, 360, 312,
 315, 331, 332, 858, 859, 240, 241, 231, 233, 52, 232, 607, 1001, 226, 238, 239, 224, 234,
 230, 227, 260, 246, 1010, 599, 799, 609, 247, 243, 245, 244, 248, 953, 597, 261, 250, 237
, 107, 185, 594, 595, 171, 172, 162, 161, 734, 156, 147, 736, 745, 128, 714, 713, 712]
```

Figura 1: Exemplo de um caminho entre as paragens 583 e 712

```
10 ?- caminho(583,732,R).
R = [583, 584, 585, 576, 941, 581, 579, 969, 944, 577, 913, 967, 501, 505, 521, 310, 538,
518, 540, 10, 543, 516, 503, 542, 541, 419, 88, 869, 989, 379, 378, 294, 281, 283, 282, 28
5, 284, 828, 796, 795, 191, 797, 208, 785, 781, 756, 762, 778, 777, 773, 774, 776, 775, 77
9, 751, 780, 732].
```

Figura 2: Exemplo de um caminho entre as paragens 583 e 732

3.2.2 Caminho entre Duas Paragens com Operadoras Escolhidas

Tendo o algoritmo base desenvolvido, esta interrogação em pouco altera o algoritmo. Altera o predicado que cria o caminho numa carreira idêntica porque tem de verificar se é possível na carreira atual prosseguir sempre utilizando sempre as Operadoras selecionadas.

```
1  % Retorna a lista de elementos um Inicio e um Fim na mesma carreira.
2  criarCaminho2(_,Fim,Fim,_,[Fim]):- !.
3
4  criarCaminho2(Carreira,Inicio,Fim,Operadoras,Adjacente):-
5      paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
6      paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
7      Id1 < IdAux,
8      Id2 is Id1 + 1,
9      paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
10     \+ memberchk(Op1, Operadoras),
```

```
    !, fail.

criarCaminho2(Carreira,Inicio,Fim,Operadoras,[Inicio|Caminho]):-
    paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
    Id1 < IdAux,
    Id2 is Id1 + 1,
    paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
    member(Op1, Operadoras),
    criarCaminho2(Carreira, Adjacente, Fim,Operadoras, Caminho).

criarCaminho2(Carreira,Inicio,Fim,Operadoras,Adjacente):-
    paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
    Id1 > IdAux,
    Id2 is Id1 - 1,
    paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
    \+ memberchk(Op1, Operadoras),
    !,fail.

criarCaminho2(Carreira,Inicio,Fim,Operadoras,[Inicio|Caminho]):-
    paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
    paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
    Id1 > IdAux,
    Id2 is Id1 - 1,
    paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
    member(Op1, Operadoras),
    criarCaminho2(Carreira, Adjacente, Fim, Operadoras,Caminho).
```

Para o restante código, o algoritmo analisado em 3.2, o predicado **caminho** será substituído por **caminho_2** onde tem em conta se em qualquer instante se selecionam Operadoras que pertençam à lista dada pelo utilizador, como se vê no seguinte excerto (apenas uma das condições do algoritmo, para exemplificar):

```
% Predicado que cria um caminho entre duas paragens, se tiver um ponto de
    juncao comum
caminho_2(Inicio,Fim,Operadoras,Caminho):-
    paragem(Carreira1,_ ,Inicio, _, _, _, _, _, Op1, _, _, _),
    member(Op1, Operadoras),
    paragem(Carreira2,_ ,Fim, _, _, _, _, _, Op2, _, _, _),
    member(Op2, Operadoras),
    carreirasAdjacentesCarreira(Carreira2,Adjs2),
    flatten(Adjs2,Help),
    member(Carreira1,Help),
    paragem(Carreira1,_,Escolhido, _, _, _, _, _, Op3, _, _, _),
    member(Op3, Operadoras),
    paragem(Carreira2,_,Escolhido, _, _, _, _, _, Op4, _, _, _),
    member(Op4, Operadoras),
    criarCaminho2(Carreira1, Inicio, Escolhido, Operadoras,Resultante1),
    criarCaminho2(Carreira2, Escolhido, Fim,Operadoras, Resultante2x),
```

```
16      removehead(Resultante2x,Resultante2),
17      append(Resultante1, Resultante2, Caminho).
```

Como podemos ver, o predicado funciona corretamente para, por exemplo, o caminho entre as paragens 183 e 499, que caso consideremos as duas operadoras que fazem parte desta carreira (ambos estão na carreira um), a 'Vimeca' e a 'SCoTTURB', o caminho é bastante mais curto do que se apenas considerarmos paragens com 'Vimeca'.

```
14 ?- caminho_2(183,499,['Vimeca'],R).
R = [183, 791, 595, 594, 180, 181, 593, 499] .

15 ?- caminho_2(183,499,['Vimeca','SCoTTURB'],R).
R = [183, 791, 595, 182, 499].
```

Figura 3: Exemplo de um caminho entre as paragens 183 e 499 utilizando uma lista de operadoras pretendidas

3.2.3 Caminho entre Duas Paragens com Operadoras Escolhidas Excluídas

Para esta interrogação foi reaproveitado o código utilizado na anterior, mas invertendo um dos predicados, o **member**, visto que agora queremos que as Operadoras **não** pertençam àquelas fornecidas pelo utilizador.

```
1  % 3. Excluir um ou mais operadores de transporte para o percurso.
2  criarCaminho3(_,Fim,Fim,_,[Fim]):- !.
3
4  criarCaminho3(Carreira,Inicio,Fim,Operadoras,Adjacente):-
5      paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _),
6      paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _),
7      Id1 < IdAux,
8      Id2 is Id1 + 1,
9      paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
10     member(Op1, Operadoras),
11     !, fail.
12
13 criarCaminho3(Carreira,Inicio,Fim,Operadoras,[Inicio|Caminho]):-
14     paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
15     paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
16     Id1 < IdAux,
17     Id2 is Id1 + 1,
18     paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
19     \+ memberchk(Op1, Operadoras),
20     criarCaminho3(Carreira, Adjacente, Fim,Operadoras, Caminho).
21
22 criarCaminho3(Carreira,Inicio,Fim,Operadoras,Adjacente):-
```

```
23      paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _),
24      paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _),
25      Id1 > IdAux,
26      Id2 is Id1 - 1,
27      paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
28      member(Op1, Operadoras),
29      !,fail.
30
31  criarCaminho3(Carreira,Inicio,Fim,Operadoras,[Inicio|Caminho]):-
32      paragem(Carreira,Id1, Inicio, _, _, _, _, _, _, _, _, _),
33      paragem(Carreira,IdAux, Fim, _, _, _, _, _, _, _, _, _),
34      Id1 > IdAux,
35      Id2 is Id1 - 1,
36      paragem(Carreira,Id2, Adjacente, _, _, _, _, _, Op1, _, _, _),
37      \+ memberchk(Op1, Operadoras),
38      criarCaminho3(Carreira, Adjacente, Fim, Operadoras,Caminho).
```

A título de exemplo, verifique-se o mesmo segmento do código apresentado anteriormente na nova versão:

```
1   % Predicado que cria um caminho entre duas paragens, se tiver um ponto de
        jun   o comum
2   caminho_3(Inicio,Fim,Operadoras,Caminho):-
3       paragem(Carreira1,_ ,Inicio, _, _, _, _, _, Op1, _, _, _),
4       \+ memberchk(Op1, Operadoras),
5       paragem(Carreira2,_ ,Fim, _, _, _, _, _, Op2, _, _, _),
6       \+ memberchk(Op2, Operadoras),
7       carreirasAdjacentesCarreira(Carreira2,Adjs2),
8       flatten(Adjs2,Help),
9       member(Carreira1,Help), % procura se a carreira original est  nas
            atingiveis pela carreira que contem o no final
10      paragem(Carreira1,_,Escolhido, _, _, _, _, _, Op3, _, _, _), % vou buscar
             esse comum
11      \+ memberchk(Op3, Operadoras),
12      paragem(Carreira2,_,Escolhido, _, _, _, _, _, Op4, _, _, _), % vou buscar
             esse comum
13      \+ memberchk(Op4, Operadoras),
14      criarCaminho3(Carreira1, Inicio, Escolhido, Operadoras,Resultante1),
15      criarCaminho3(Carreira2, Escolhido, Fim,Operadoras, Resultante2x), % fica
             aqui com um elemento a mais, tira-se
16      removehead(Resultante2x,Resultante2),
17      append(Resultante1, Resultante2, Caminho).
```

É visível que é idêntico ao anterior, só muda a condição de inclusão da carreira, que depende se é ou não das operadoras pretendidas e, como tal, os resultados são apresentados invertidos relativamente aos anteriores para o mesmo caso.

```
19 ?- caminho_3(183,499,['SCoTTURB'],R).
R = [183, 791, 595, 594, 180, 181, 593, 499] .

20 ?- caminho_3(183,499,[],R).
R = [183, 791, 595, 182, 499].
```

Figura 4: Exemplo de um caminho entre as paragens 183 e 499 utilizando uma lista de operadoras que não são pretendidas

3.2.4 Paragem com Maior Número de Carreiras num Caminho entre Duas Paragens

Esta interrogação não envolve o algoritmo em si, pelo que pode ser invocada a versão base para descobrir o caminho entre duas paragens. No entanto, surge a necessidade de retirar o número de carreiras a que todas as paragens nesse caminho escolhido pertencem e, desse número de carreiras, retirar a(s) paragem(ns) com o número maior de carreiras.

```
carreiras(Paragem,Total):-
    findall(Carreira, (paragem(Carreira,_,Paragem, _, _, _, _, _, _, _, _, _)
        ),
    Lista),
    length(Lista,Total).

% Verifica para todas as paragens qual aquela que pertence ao maior n mero
    de carreiras e retorna-o.
calculaMax(T,Max):-
    maplist(carreiras, T, MaxAux),
    max_list(MaxAux, Max).

caminho_4(Inicio,Fim,X/Maior) :-
    caminho(Inicio,Fim,Caminho),
    calculaMax(Caminho,Maior),
    member(X,Caminho),
    carreiras(X,Maior).
```

Como podemos ver no exemplo seguinte, para o caminho entre a paragem 183 e a paragem 185, há 3 paragens que têm o maior número de carreiras igual, que o algoritmo deteta. Retorna sobre a forma de GID/MAX, como se vê na definição do predicado **caminho_4**.

```
20 ?- caminho_4(183,185,R).
R = 595/7 ;
R = 594/7 ;
R = 185/7.
```

Figura 5: Máximo de carreiras para as paragens no caminho entre as paragens 183 e 185

3.2.5 Caminho entre Duas Paragens com o Menor Número de Paragens

Como esta interrogação não exige um caminho com condições especiais, utiliza o predicado de **caminho** definido em 3.2, no entanto, utiliza o predicado de **findall** para encontrar todos os caminhos possíveis entre duas paragens, escolhendo dentro deles qual o menor utilizando o predicado **menorLista**.

```
% Predicado que calcula qual a menor lista num conjunto de listas.
lengths([],[]).
lengths([H|T], [LH|LengthsT]) :-
    length(H, LH),
    lengths(T, LengthsT).

tamanhoMenor(ListOfLists, Min) :-
    lengths(ListOfLists, Lengths),
    min_list(Lengths, Min).

menorLista(ListOfLists, Min/Len) :-
    tamanhoMenor(ListOfLists, Len),
    member(Min, ListOfLists),
    length(Min, Len).

menorCaminhoParagens(Inicio,Fim,Caminho/D):-
    findall(R,(caminho(Inicio,Fim,R)),L),
    menorLista(L,Caminho/D).
```

Utilizando o mesmo raciocínio do problema anterior, este apresenta os resultados com uma lista de paragens seguido do seu comprimento, como se vê no exemplo seguinte.

```
21 ?- menorCaminhoParagens(183,185,R).
R = [183, 791, 595, 182, 499, 593, 181, 180|...]/10.
```

Figura 6: Caminho mais curto (em nº de paragens) entre as paragens 183 e 185

3.2.6 Caminho entre Duas Paragens com a Menor Distância Percorrida

Esta interrogação já foi um bocado mais complicada de implementar, apesar de utilizar o mesmo algoritmo base que a da secção anterior. Primeiro, era necessário obter todos os caminhos entre duas paragens com o **findall**, depois era necessário transformar todos esses caminhos numa lista com apenas as coordenadas (Latitude e Longitude) e, usando essa lista, calcular a distância euclidiana entre os vários pontos consecutivos na lista. Após obter a distância euclidiana, era necessário somá-las e, por fim, obter a menor distância de todos os caminhos.

```
% Predicado que calcula a distancia euclideana entre 2 pontos
d([P|Ps], [Q|Qs], D) :-
    soma_dif_sq(Ps, Qs, (P-Q)^2, R),
    D is sqrt(R).

soma_dif_sq([], [], V, V).
soma_dif_sq([P|Ps], [Q|Qs], V0, V+V0) :-
    soma_dif_sq(Ps, Qs, (P-Q)^2, V).

% Predicado que guarda a lista de coordenadas de todas as paragens de um
    caminho
caminho2ListaPontos([],[]).

caminho2ListaPontos([H|T], [[P1,P2]|R]):-
    paragem(_,_ ,H, P1, P2, _, _, _, _, _, _, _),
    caminho2ListaPontos(T,R),
    !.

distCaminho(_,[],0).

distCaminho(Atual,[H|T],R):-
    d(Atual,H,Soma),
    distCaminho(H,T,Soma2),
    R is Soma + Soma2.

% Predicado que calcula qual a menor lista num conjunto de listas.
dists([],[]).
dists([[[H1,H2]|R]|T], [LH|Dists]) :-
    distCaminho([H1,H2],[[H1,H2]|R],LH),
    dists(T, Dists).

distMenor(ListOfLists, Min) :-
    dists(ListOfLists, Lengths),
    min_list(Lengths, Min).

menorDistLista([[[H1,H2]|R]|T],[[H3,H4]|R2]/Len) :-
    distMenor([[[H1,H2]|R]|T], Len),
    member([[H3,H4]|R2],[[[H1,H2]|R]|T]),
    distCaminho([H3,H4],[[H3,H4]|R2],Len).
```

```
39
40 menorCaminhoDist(Inicio,Fim,Caminho/P):-
41     findall(R,(caminho(Inicio,Fim,R)),L),
42     maplist(caminho2ListaPontos, L, LPontos),
43     menorDistLista(LPontos,Pontos/P),
44     member(Caminho,L),
45     caminho2ListaPontos(Caminho,Pontos).
```

Utilizando o mesmo raciocínio do problema anterior, este apresenta os resultados com uma lista de paragens seguido da distância desde a paragem inicial até à final, como se vê no exemplo seguinte.

```
25 ?- menorCaminhoDist(183,185,R).
R = [183, 791, 595, 182, 499, 593, 181, 180|...]/6315.847800959968 .
```

Figura 7: Caminho mais curto (em distância física) entre as paragens 183 e 185

3.2.7 Caminho entre Duas Paragens onde os Abrigos têm Publicidade

Assim como as interrogações iniciais, tendo o algoritmo base completo não é necessário mudar muito para que respeite a condição imposta por esta em específico. Basta que os pontos de ligação respeitem esta condição e, se não verificarem, não é realizado o caminho. Veja-se o seguinte excerto de código (visto que os restantes sofrem alterações semelhantes relativamente ao original):

```
1  % Predicado que cria um caminho entre duas paragens, se tiver um ponto de
       juncao comum
2  caminho_7(Inicio,Fim,Caminho):-
3      paragem(Carreira1,_, Inicio, _, _, _, _,  Yes , _, _, _, _),
4      paragem(Carreira2,_ ,Fim, _, _, _, _,  Yes , _, _, _, _),
5      carreirasAdjacentesCarreira(Carreira2,Adjs2),
6      flatten(Adjs2,Help),
7      member(Carreira1,Help),
8      paragem(Carreira1,_,Escolhido, _, _, _, _,  Yes , _, _, _, _),
9      paragem(Carreira2,_,Escolhido, _, _, _, _,  Yes , _, _, _, _),
10     criarCaminho(Carreira1, Inicio, Escolhido, Resultante1),
11     criarCaminho(Carreira2, Escolhido, Fim, Resultante2x),
12     removehead(Resultante2x,Resultante2),
13     append(Resultante1, Resultante2, Caminho).
```

Para o exemplo da paragem 499 da carreira 1, esta não possui publicidade no seu abrigo, portanto, é condição suficiente para falhar. Se considerarmos a paragem imediatamente antes, a 499, já a verifica e, portanto, é possível realizar o trajeto.

```
26 ?- caminho_7(183,593,R).
false.

27 ?- caminho_7(183,499,R).
R = [183, 791, 595, 182, 499].
```

Figura 8: Caminho desde a paragem 183 até 593, sem publicidade, e 499, com publicidade

3.2.8 Caminho entre Duas Paragens com Paragens Abrigadas

Muito semelhante à anterior, muda apenas o campo que é feito o teste, isto é, é verificado se é uma paragem que tem abrigo e, se for, é elegível para o percurso. Salienta-se o seguinte excerto de código para análise (visto que os restantes sofrem alterações semelhantes relativamente ao original):

```
% Predicado que cria um caminho entre duas paragens, se tiver um ponto de
    juncao comum
caminho_8(Inicio,Fim,Caminho):-
    paragem(Carreira1,_, Inicio, _, _, _, Tipo, _, _, _, _, _),
    Tipo \= Sem␣Abrigo ,
    paragem(Carreira2,_ ,Fim, _, _, _, Tipo2, _, _, _, _, _),
    Tipo2 \= Sem␣Abrigo ,
    carreirasAdjacentesCarreira(Carreira2,Adjs2),
    flatten(Adjs2,Help),
    member(Carreira1,Help),
    paragem(Carreira1,_,Escolhido, _, _, _, Tipo3, _, _, _, _, _),
    Tipo3 \= Sem␣Abrigo ,
    paragem(Carreira2,_,Escolhido, _, _, _, Tipo4, _, _, _, _, _),
    Tipo4 \= Sem␣Abrigo ,
    criarCaminho(Carreira1, Inicio, Escolhido, Resultante1),
    criarCaminho(Carreira2, Escolhido, Fim, Resultante2x),
    removehead(Resultante2x,Resultante2),
    append(Resultante1, Resultante2, Caminho).
```

Considere-se o cenário de teste para a carreira 2 desta vez. A paragem 147 é do tipo "Sem Abrigo", ou seja, não é elegível. Se considerarmos uma outra paragem na mesma carreira que é a 161, esta já é elegível e, portanto, o caminho já se pode realizar. Salienta-se que apesar de passar no 147, é um caminho válido porque o indivíduo só sairá na paragem 161, o estado das restantes paragens não é relevante.

```
30 ?- caminho_8(745,161,R).
R = [745, 736, 147, 156, 734, 161].

31 ?- caminho_8(745,147,R).
false.
```

Figura 9: Caminho desde a paragem 745 até à 161, com algum tipo de abrigo, e à 499, sem abrigo

3.2.9 Caminho entre Duas Paragens com Paragens Intermédias

Esta interrogação nada traz de novo em relação ao algoritmo original, pelo que é apenas uma questão de percorrer uma lista de paragens e tentar construir caminhos entre todos eles.

```
removeLast(L, T):- reverse(L,Rev),removehead(Rev,Rem), reverse(Rem,T).

caminho_9(Inicio,Fim,[],Caminho):-caminho(Inicio,Fim,Caminho).

caminho_9(Inicio,Fim,[Intermedio|T],Caminho) :-
    caminho(Inicio,Intermedio,Parcial1),
    removeLast(Parcial1,Parcial),
    caminho_9(Intermedio,Fim,T,X),
    append(Parcial,X,Caminho).
```

Como é possível verificar no seguinte teste, o algoritmo funciona pelo que passa em todos os pontos pedidos:

```
33 ?- caminho_9(97,526,[178,250],R).
R = [97, 98, 99, 707, 717, 716, 726, 727, 725, 132, 722, 723, 130, 731, 749, 750, 751, 775
, 776, 777, 778, 748, 757, 762, 756, 782, 786, 784, 772, 783, 694, 213, 259, 222, 109, 108
, 257, 258, 276, 277, 287, 1004, 290, 289, 288, 388, 437, 884, 397, 396, 274, 273, 296, 31
6, 305, 297, 306, 299, 303, 318, 311, 304, 319, 302, 1005, 826, 833, 317, 320, 838, 836, 8
35, 837, 842, 841, 810, 811, 802, 804, 632, 803, 800, 809, 808, 805, 801, 1009, 236, 816,
813, 817, 692, 693, 178, 947, 792, 710, 807, 818, 823, 952, 954, 1002, 977, 986, 983, 354,
 353, 863, 856, 857, 367, 333, 846, 845, 330, 364, 33, 32, 60, 61, 64, 63, 62, 58, 57, 59,
 654, 655, 56, 491, 490, 458, 457, 335, 363, 344, 345, 343, 346, 342, 341, 85, 86, 347, 33
9, 352, 351, 323, 313, 360, 312, 315, 331, 332, 858, 859, 240, 241, 231, 233, 52, 232, 607
, 1001, 226, 238, 239, 224, 234, 230, 227, 260, 246, 1010, 599, 799, 609, 247, 243, 245, 2
44, 248, 953, 597, 261, 250, 261, 597, 953, 605, 606, 609, 82, 604, 628, 39, 50, 599, 40,
622, 51, 38, 620, 45, 602, 601, 860, 861, 359, 349, 29, 646, 642, 30, 17, 643, 20, 36, 638
, 637, 361, 362, 37, 26, 27, 28, 641, 635, 679, 688, 675, 72, 75, 671, 657, 70, 526]
```

Figura 10: Caminho desde a paragem 97 até à 526, passando nas paragens 178 e 250

3.3 Algoritmo com Pesquisa Não-Informada

Para esta estratégia de pesquisa, grande parte dos pontos, pré-definidos no início da secção principal, não podem ser respeitados em virtude de ser não informada e, portanto, será inerentemente mais ineficiente e, infelizmente, não funciona para problemas demasiadamente complexos, que é o caso deste.

Para este tipo de pesquisa, foi desenvolvido uma estratégia muito conhecida e dada nas aulas teórico-praticas, o algoritmo de **primeiro em largura**, visto que é o candidato que mais se adequa a este problema porque consegue-se de imediato eliminar caminhos que não são relevantes. É, também, **bi-direcional**.

O objetivo do algoritmo é colocar numa lista de espera todos os nós adjacentes ao nó atual e visitar, por ordem, todos os nós apartir retirando um elemento de cada vez da lista de espera.

```
% Predicado que verifica se 2 paragens sao adjacentes
adjacentes(X,Y) :- adjacente(X,Y) ; adjacente(Y,X).

% Realiza-se a invoca  o de um breadth-first para resolver o problema.
caminho2(Inicio,Fim,Caminho) :-
    bfs([[Inicio]],Fim, CaminhoAux),
    reverse(CaminhoAux,Caminho).

% Se estiver na paragem final termina.
bfs([[Fim|Caminho]|_],Fim, [Fim|Caminho]).

% Se n o, extende o caminho atual por todas as paragens que o conseguem
    fazer e coloca na fila de espera para serem verificados, reiterando pelos
    novos caminhos encontrados.
bfs([Caminho1|Caminhos], Fim, Caminho) :-
    extende(Caminho1, NovosCaminhos),
    append(Caminhos, NovosCaminhos, Caminhos1),
    bfs(Caminhos1, Fim, Caminho).

% Predicado que extende o caminho utilizando todas as paragens que n o
    estejam dentro do caminho.
extende([Paragem|Caminho], NovosCaminhos) :-
    findall([Paragem2, Paragem|Caminho],(adjacentes(Paragem, Paragem2),\+
        memberchk(Paragem2,[Paragem|Caminho])),NovosCaminhos),!.

extende(_,[]).
```

3.3.1 Caminho entre Duas Paragens

Esta interrogação é a mais simples que se pode fazer com o algoritmo base terminado, visto que apenas o utiliza diretamente. Por exemplo:

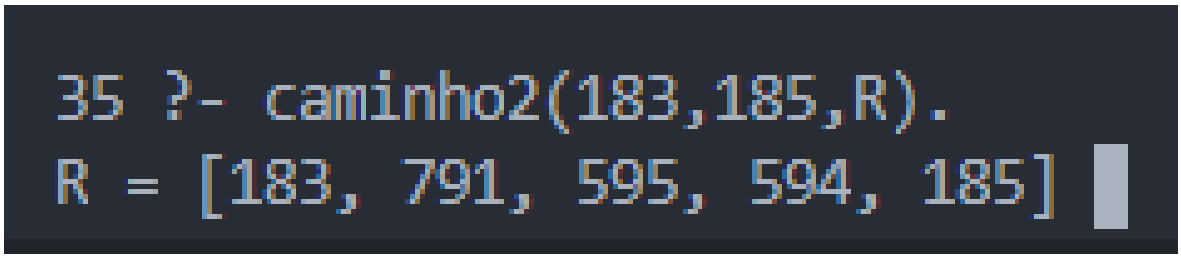

Figura 11: Caminho desde a paragem 183 até à 185

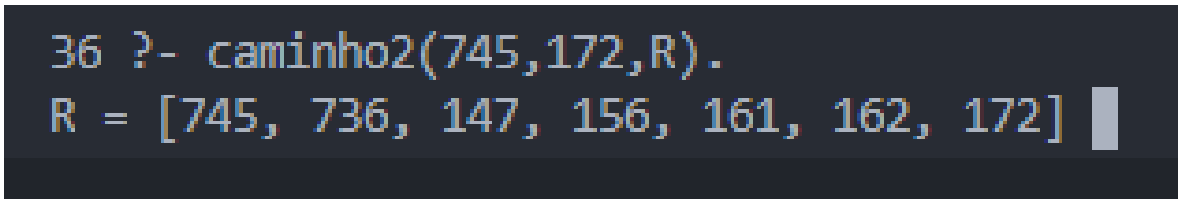

Figura 12: Caminho desde a paragem 745 até à 172

3.3.2 Caminho entre Duas Paragens com Operadoras Escolhidas

Semelhante ao que acontecia na , apenas se mudam alguns predicados relativamente ao algoritmo original para garantir que a Operadora da paragem adjacente é, de facto, pertencente a uma das Operadoras indicadas pelo utilizador, como se verifica no seguinte excerto de código:

```
caminho2_2(Inicio,Fim,Operadoras,Caminho) :-
    paragem(_,_ ,Inicio, _, _, _, _, _, Op1, _, _, _),
    paragem(_,_ ,Fim, _, _, _, _, _, Op2, _, _, _),
    member(Op1, Operadoras),
    member(Op2, Operadoras),
    bfs2([[Inicio]],Fim, Operadoras,CaminhoAux),
    reverse(CaminhoAux,Caminho).

    (...)

extende2([Paragem|Caminho], Operadoras, NovosCaminhos) :-
    findall([Paragem2, Paragem|Caminho],(adjacentes(Paragem, Paragem2),\+
        memberchk(Paragem2,[Paragem|Caminho]),
    paragem(_,_ ,Paragem2, _, _, _, _, _, Op1, _, _, _), member(Op1,
        Operadoras)),NovosCaminhos),!.
```

Como vemos no seguinte exemplo, é possível estabelecer um caminho entre as paragens 745 e 734 da carreira dois usando apenas a operadora 'Vimeca', no entanto, não o é possível fazer com a operadora 'SCoTTURB' entre as paragens 183 e 182, visto que a paragem 183 pertence à operadora 'Vimeca'.

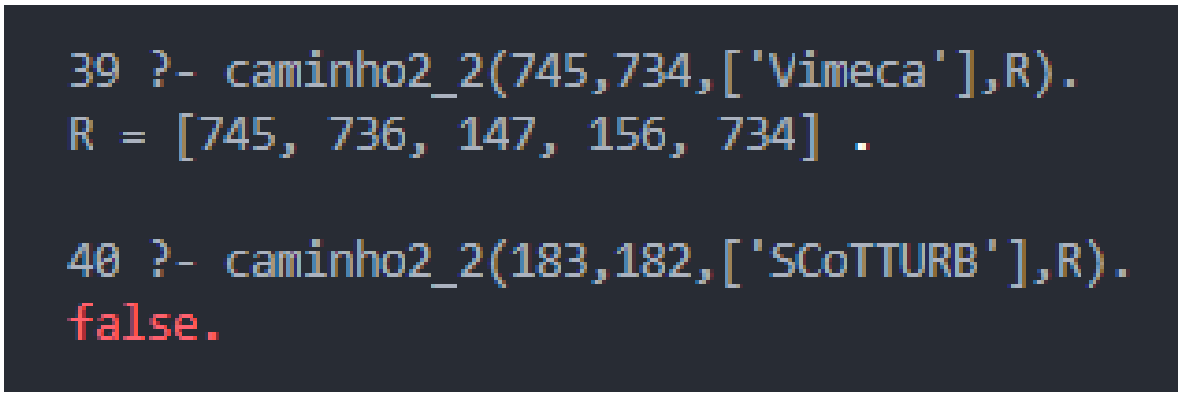

Figura 13: Caminho desde a paragem 745 até à 734 e da 183 à 182 utilizando operadoras distintas

3.3.3 Caminho entre Duas Paragens com Operadoras Escolhidas Excluídas

Para esta interrogação, verificamos que é exatamente o contrário da anterior e, portanto, queremos saber quando **não** são operadoras que o utilizador mencionou que não pretendia para o seu percurso, como se vê pelo excerto de código seguinte:

```
caminho2_3(Inicio,Fim,Operadoras,Caminho) :-
    paragem(_,_ ,Inicio, _, _, _, _, _, Op1, _, _, _),
    paragem(_,_ ,Fim, _, _, _, _, _, Op2, _, _, _),
    \+ memberchk(Op1, Operadoras),
    \+ memberchk(Op2, Operadoras),
    bfs3([[Inicio]],Fim, Operadoras,CaminhoAux),
    reverse(CaminhoAux,Caminho).

(...)

extende3([Paragem|Caminho], Operadoras, NovosCaminhos) :-
    findall([Paragem2, Paragem|Caminho],(adjacentes(Paragem, Paragem2),\+
        memberchk(Paragem2,[Paragem|Caminho]),
    paragem(_,_ ,Paragem2, _, _, _, _, _, Op1, _, _, _),\+ memberchk(Op1,
        Operadoras)),NovosCaminhos),!.
```

Como é possível ver pela figura seguinte e pegando no caso da interrogação anterior das paragens 183 e 182, não é possível estabelecer um caminho entre estas duas paragens sem utilizar a Operadora 'SCoTTURB'.

```
41 ?- caminho2_3(183,182,['SCoTTURB'],R).
false.

42 ?- caminho2_3(183,182,[],R).
R = [183, 791, 595, 182]
```

Figura 14: Caminho desde a paragem 183 até à 182 com paragens excluídas

3.3.4 Paragem com Maior Número de Carreiras num Caminho entre Duas Paragens

O procedimento é exatamente igual ao apresentado em 3.2.4, porque a interrogação é a mesma e o processo para lidar com ela não depende do algoritmo escolhido para determinar o caminho entre duas paragens.

```
% Retorna o numero de carreiras cuja paragem est  localizada
carreiras(Paragem,Total):-
    findall(Carreira, (paragem(Carreira,_,Paragem, _, _, _, _, _, _, _, _, _)
        ),
    Lista),
    length(Lista,Total).

% Verifica para todas as paragens qual aquela que pertence ao maior n mero
    de carreiras e retorna-o.
calculaMax(T,Max):-
    maplist(carreiras, T, MaxAux),
    max_list(MaxAux, Max).

caminho2_4(Inicio,Fim,X/Maior) :-
    caminho2(Inicio,Fim,Caminho),
    calculaMax(Caminho,Maior),
    member(X,Caminho),
    carreiras(X,Maior).
```

Utilizando, portanto, o mesmo exemplo em 3.2.4, verificamo que o resultado é exatamente igual.

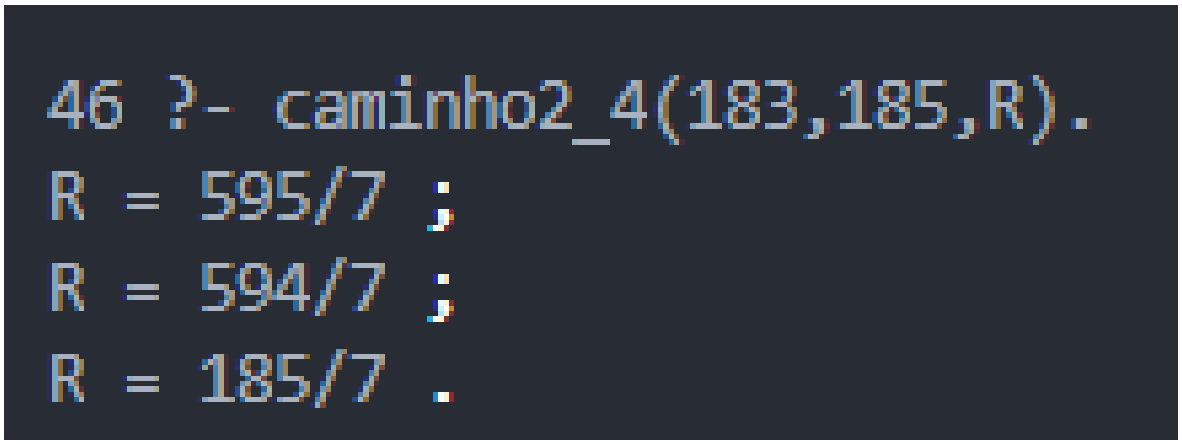

Figura 15: Máximo de carreiras para as paragens no caminho entre as paragens 183 e 185

3.3.5 Caminho entre Duas Paragens com o Menor Número de Paragens

Também utiliza o mesmo algoritmo que em 3.2.5 porque, tal como a alínea anterior, não depende do algoritmo para encontrar o caminho, apesar de agora ser utilizado este novo algoritmo, apenas os encontra todos e desses todos obtém aquele que é o melhor, tal como o analisado na referência fornecida.

```
% Predicado que calcula qual a menor lista num conjunto de listas.
lengths([],[]).
lengths([H|T], [LH|LengthsT]) :-
    length(H, LH),
    lengths(T, LengthsT).

tamanhoMenor(ListOfLists, Min) :-
    lengths(ListOfLists, Lengths),
    min_list(Lengths, Min).

menorLista(ListOfLists, Min/Len) :-
    tamanhoMenor(ListOfLists, Len),
    member(Min, ListOfLists),
    length(Min, Len).

menorCaminhoParagens(Inicio,Fim,Caminho/D):-
    findall(R,(caminho2(Inicio,Fim,R)),L),
    menorLista(L,Caminho/D).
```

Para o caso de testes, considere-se o caminho entre as paragens 674 e 658, que é um caminho bastante curto no que toca ao número de paragens, que se espera que o resultado dê exatamente o que se espera.

```
2 ?- menorCaminhoParagens(674,658,R).
R = [674, 73, 658]/3.
```

Figura 16: Caminho mais curto entre 674 e 658, quanto ao número de paragens

3.3.6 Caminho entre Duas Paragens com a Menor Distância Percorrida

Novamente, este procedimento é também exatamente igual ao 3.2.6, visto que também não depende do algoritmo do caminho para encontrar a melhor solução, simplesmente chama o algoritmo atual de caminho com o predicado **findall** para os obter a todos e, do total, obtém aquele que é o melhor.

```
% Predicado que calcula a distancia euclideana entre 2 pontos
d([P|Ps], [Q|Qs], D) :-
    soma_dif_sq(Ps, Qs, (P-Q)^2, R),
    D is sqrt(R).

soma_dif_sq([], [], V, V).
soma_dif_sq([P|Ps], [Q|Qs], V0, V+V0) :-
    soma_dif_sq(Ps, Qs, (P-Q)^2, V).

% Predicado que guarda a lista de coordenadas de todas as paragens de um
    caminho
caminho2ListaPontos([],[]).

caminho2ListaPontos([H|T], [[P1,P2]|R]):-
    paragem(_,_ ,H, P1, P2, _, _, _, _, _, _, _),
    caminho2ListaPontos(T,R),
    !.

distCaminho(_,[],0).

distCaminho(Atual,[H|T],R):-
    d(Atual,H,Soma),
    distCaminho(H,T,Soma2),
    R is Soma + Soma2.

% Predicado que calcula qual a menor lista num conjunto de listas.
dists([],[]).
dists([[[H1,H2]|R]|T], [LH|Dists]) :-
    distCaminho([H1,H2],[[H1,H2]|R],LH),
    dists(T, Dists).

distMenor(ListOfLists, Min) :-
    dists(ListOfLists, Lengths),
    min_list(Lengths, Min).

menorDistLista([[[H1,H2]|R]|T],[[H3,H4]|R2]/Len) :-
```

```
36      distMenor([[[H1,H2]|R]|T], Len),
37      member([[H3,H4]|R2],[[[H1,H2]|R]|T]),
38      distCaminho([H3,H4],[[H3,H4]|R2],Len).
```

Como podemos ver pelo exemplo seguinte e, utilizando o mesmo exemplo da alínea anterior, obtemos o mesmo caminho com uma distância bastante curta.

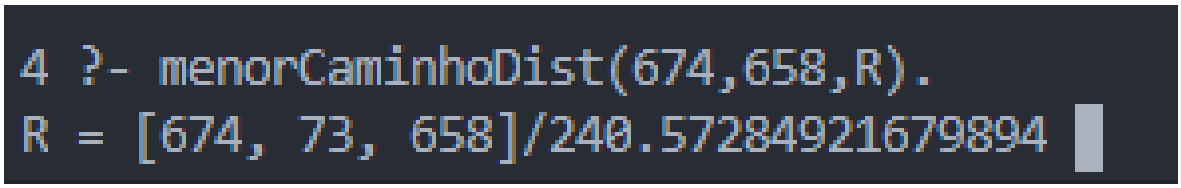

Figura 17: Caminho mais curto entre 674 e 658, quanto à distância física

3.3.7 Caminho entre Duas Paragens onde os Abrigos têm Publicidade

Para esta interrogação não era necessário nada mais que alterar os predicados para garantir esta condição, isto é, só passa para a próxima paragem se esta efetivamente tiver um abrigo com publicidade, como se vê no excerto de código que se segue.

```
caminho2_7(Inicio,Fim,Caminho) :-
    paragem(_,_ ,Inicio, _, _, _, _,  Yes , _, _, _, _),
    paragem(_,_ ,Fim, _, _, _, _,  Yes , _, _, _, _),
    bfs7([[Inicio]],Fim,CaminhoAux),
    reverse(CaminhoAux,Caminho).

    (...)

extende7([Paragem|Caminho], NovosCaminhos) :-
    findall([Paragem2, Paragem|Caminho],(adjacentes(Paragem, Paragem2),\+
        memberchk(Paragem2,[Paragem|Caminho]),
    paragem(_,_ ,Paragem2, _, _, _, _,  Yes , _, _, _, _)),NovosCaminhos),!.
```

Como é possível ver no exemplo que se segue, a paragem 666 não possui publicidade enquanto a 73 possui, pelo que não é possível criar um caminho para a paragem 666.

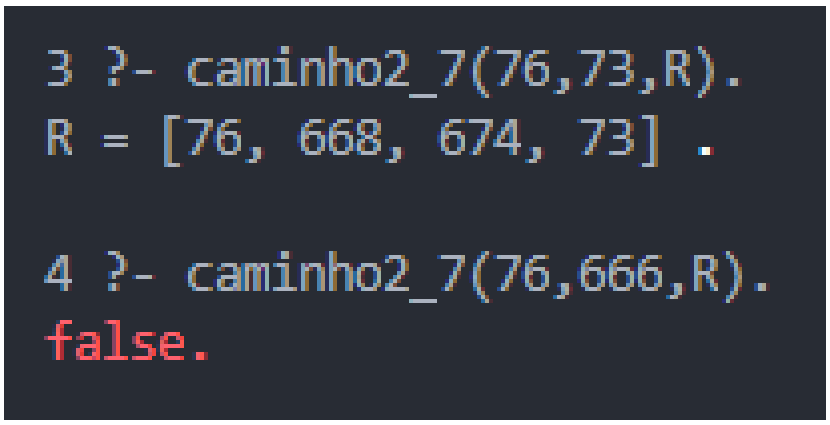

Figura 18: Caminho entre 76 e 73, com publicidade, e 666, sem publicidade

3.3.8 Caminho entre Duas Paragens com Paragens Abrigadas

Mais uma vez, não é alterado essencialmente nada no algoritmo original, apenas se adicionam as seguintes condições para garantir que todas as paragem adjacentes possuem abrigo, assim como as iniciais e finais, como se verifica no excerto de código em seguida.

```
caminho2_8(Inicio,Fim,Caminho) :-
    paragem(_,_ ,Inicio, _, _, _, Tipo, _, _, _, _, _),
    paragem(_,_ ,Fim, _, _, _, Tipo2, _, _, _, _, _),
    Tipo \= Sem␣Abrigo ,
    Tipo2 \= Sem␣Abrigo ,
    bfs8([[Inicio]],Fim,CaminhoAux),
    reverse(CaminhoAux,Caminho).

    (...)

extende8([Paragem|Caminho], NovosCaminhos) :-
    findall([Paragem2, Paragem|Caminho],(adjacentes(Paragem, Paragem2),\+
        memberchk(Paragem2,[Paragem|Caminho]),
    paragem(_,_ ,Paragem2, _, _, _, Tipo, _, _, _, _, _), Tipo \= Sem␣Abrigo
        ),NovosCaminhos),!.
```

A título de exemplo considere-se agora o trajeto entre as paragens 668 e 658 e entre 668 e 666. A paragem 666 não possui abrigo e a paragem 658 possui, pelo que as intermédias todas possuem. Portanto, no segundo caso terá de dar falso e no primeiro terá de dar verdadeiro quando questionamos o motor de inferência, que se verifica na imagem seguinte.

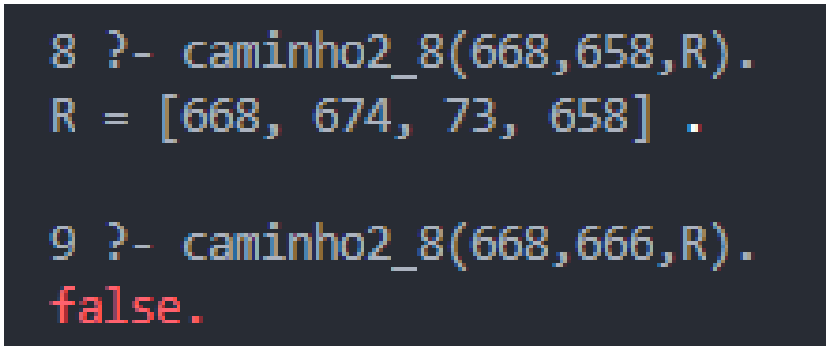

Figura 19: Caminho entre 668 e 658, com abrigo, e 666, sem abrigo

3.3.9 Caminho entre Duas Paragens com Paragens Intermédias

Tal como em 3.2.9, o predicado é muito semelhante porque não depende do algoritmo escolhido para o caminho, apenas se realizam caminhos sucessivos entre os vários elementos da lista escolhida pelo utilizador e guarda-se o resultado final.

```
removehead([_|T], T).
removeLast(L, T):- reverse(L,Rev),removehead(Rev,Rem), reverse(Rem,T).

caminho2_9(Inicio,Fim,[],Caminho):-caminho2(Inicio,Fim,Caminho).

```

```
6  caminho2_9(Inicio,Fim,[Intermedio|T],Caminho) :-
7      caminho2(Inicio,Intermedio,Parcial1),
8      removeLast(Parcial1,Parcial),
9      caminho2_9(Intermedio,Fim,T,X),
10     append(Parcial,X,Caminho).
```

Como podemos ver no exemplo que se segue, todos os elementos da lista aparecem corretamente no caminho podendo ver que o predicado de facto faz o que é pretendido.

```
10 ?- caminho2_9(183,185,[182,181],R).
R = [183, 791, 595, 182, 181, 180, 594, 185]
```

Figura 20: Caminho entre 183 e 185, com as paragens intermédias 182 e 181

4 Conclusões e Sugestões

4.1 Análise de Tempos de Execução

Na tabela que se apresenta de seguida, verificamos para os distintos cenários os vários tempos obtidos com ambos os algoritmos desenvolvidos.

Paragem Inicial	Paragem Final	Tamanho do Caminho - Não Informada	Tempo (segundos) - Não Informada	Inferências - Não Informada	Tamanho do Caminho - Informada	Tempo (segundos) - Informada	Inferências - Informada
183	181	5	0.030	412,915	7	0.000	34
299	88	10	12.312	241,976,906	15	0.000	74
97	725	7	0.003	13619	9	0.000	44
629	528	17	0.001	371	17	0.000	84
637	635	8	0.125	2,173,376	9	0.000	44
16	342	6	0.016	126,427	14	0.000	69
459	681	20	0.001	441	20	0.000	99
712	342	-	-	-	80	0.003	12,402
583	712	-	-	-	199	0.007	62,087
97	526	-	-	-	251	0.017	137,247

Figura 21: Tabela de comparação de tempos/inferências para os dois algoritmos

De forma geral, podemos concluir que é uma vitória significativa para a pesquisa informada, porém, não é possível testar na sua totalidade em virtude do algoritmo não-informado não aguentar tantos dados como os que são necessários para este exercício.

Saltam, no entanto, algumas conclusões sobre os algoritmos desenvolvidos:

- O algoritmo informado produz caminhos relativamente mais longos que os da não-informada;
- O algoritmo informado é muito superior em termos de eficiência (todos os pontos que não são triviais para o não informado este realiza-os em 0.000 segundos), os pontos a tracejado significam que a execução não termina;
- O algoritmo não-informado é possível ser feito melhor, visto que há uma alínea que o informado realiza trivialmente (74 inferências) enquanto que este precisa de mais de 241 milhões de inferências. Isso mostra que ele percorre caminhos totalmente absurdos e que nunca darão a solução;
- O algoritmo informado consegue realizar caminhos bastante complexos e compridos sem grandes problemas (aproximadamente 137 mil inferências), ao contrário do não-informado.

4.2 Observações Finais

Durante a realização do trabalho foi possível conhecer melhor o funcionamento da programação simbólica, assim como métodos de representação de grafos como predicados para introduzir aos algoritmos de pesquisa informada e não-informada.

Foi dada a oportunidade de perceber as diferenças entre pesquisas informadas e pesquisas não informadas, assim como a sua implementação num problema real em contexto real, tal como se observa no quadro resumo seguinte.

Algoritmo	Completo	Otimal	Tempo	Espaço	Escalável para outros problemas
Breadth-First (Primeiro em Largura)	Sim	Se as paragens estiverem num espaço amostral relativamente curto	O(b^d)	O(b^d)	Sim
Meu Algoritmo	Sim	Se as paragens pertencerem à mesma carreira	O(b)	O(b)	Não

Figura 22: Tabela de comparação dos dois algoritmos utilizados

Em suma, foi um projeto com grande potencial de exploração criativa que penso que foram atingidos os objetivos propostos e, para além do contexto de Sistemas de Representação de Conhecimento e Raciocínio, foi uma enorme mais valia para a formação de mecanismos de raciocínio gerais e incentivar olhar para problemas em prismas diferentes, porque nunca se sabe se só há apenas uma solução.

Resta o desafio de melhorar os algoritmos propostos, visto que há bastantes pontos onde é possível criar uma melhor solução ou utilizar uma heurística diferente de resolução.

5 Referências

5.1 Referências Bibliográficas

Russell and Norvig (2009). Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594.

Ivan Bratko (2000), PROLOG: Programming for Artificial Intelligence, 3rd Edition, AddisonWesley Longman Publishing Co., Inc

5.2 Referências Eletrónicas

http://www.cse.unsw.edu.au/ billw/Justsearch.html - Algoritmos de pesquisa em grafos

http://cee.uma.pt/edu/iia/acetatos/iia-Procura%20Informada.pdf - Procura Informada e Não Informada

https://www.swi-prolog.org/pldoc/doc$_for?object = manual - Documentação do SWI - Prolog$

A Anexo

A.1 Código da Interpretação dos Ficheiros de Dados

Para os predicados de *paragem*:

```
import pandas as pd
import numpy as np
import os.path

def isNumber(x):
    if not x.strip().replace( - ,   ).replace( + ,   ).replace( . ,   ).
        isdigit():
        return False
    try:
         float(x)
    except ValueError:
         return False
    return True

def csv2predicado(id):
    df = pd.read_csv("./CSV/carreira_" + str(id) + ".csv", encoding= utf-8 )

    df.dropna(inplace=True) # remove as entradas nulas, inplace significa
        para nao gerar um objeto novo mas aplicar ao df
    df.drop_duplicates(inplace=True)

    nome = "povoamento2.pl"
    if os.path.exists(nome):
        modo =  a
    else:
        modo =  w
    f = open(nome,modo,encoding= utf-8 )

    print("\n%% CARREIRA " + str(id) + "\n", file = f)
    count = 1
    for i in range(len(df)):
        predicado =  paragem(  + str(id) + ", " + str(count) + ", "
        for j in range(11):
            if(j!=7):
                valor = str(df.iloc[i,j])
                if not isNumber(valor): # se nao for um float/int
                    valor =  \   + valor +  \
                predicado = predicado + valor + ", "
        predicado = predicado[:-2]
        predicado +=  ).
        print(predicado,file=f)
        count += 1

def ler_Todos():
```

```
    for i in range(777):
        if os.path.isfile("./CSV/carreira_" + str(i) + ".csv"):
            csv2predicado(i)

ler_Todos()
```

Para os predicados de *adjacente*:

```
import pandas as pd
import numpy as np
import os.path

def csv2adjacentes(id):

    df = pd.read_csv("./CSV/carreira_" + str(id) + ".csv", encoding= utf-8 )

    df.dropna(inplace=True) # remove as entradas nulas, inplace significa
        para nao gerar um objeto novo mas aplicar ao df
    df.drop_duplicates(inplace=True)

    nome = "adjacentes.pl"
    if os.path.exists(nome):
        modo =  a
    else:
        modo =  w
    f = open(nome,modo,encoding= utf-8 )
    ant = str(df.iloc[0,0])

    print("\n%% CARREIRA " + str(id) + "\n", file = f)

    for i in range(1,len(df)):
        valor = str(df.iloc[i,0])
        predicado =  adjacente(  + ant + ", " + valor + ")."
        print(predicado,file=f)
        ant = valor

def ler_Todos():
    for i in range(777):
        if os.path.isfile("./CSV/carreira_" + str(i) + ".csv"):
            csv2adjacentes(i)

ler_Todos()
```

A.2 Resultado Final da Interpretação dos Ficheiros de Dados

```
%% CARREIRA 1

paragem(1, 1, 183, -103678.36, -96590.26, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas').
paragem(1, 2, 791, -103705.46, -96673.6, 'Bom', 'Aberto dos Lados', 'Yes', 'Vimeca', 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas').
paragem(1, 3, 595, -103725.69, -95975.2, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 354, 'Rua Manuel Teixeira Gomes', 'Carnaxide e Queijas').
paragem(1, 4, 182, -103746.76, -96396.66, 'Bom', 'Fechado dos Lados', 'Yes', 'SCoTTURB', 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas').
paragem(1, 5, 499, -103758.44, -94393.36, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 300, 'Avenida dos Cavaleiros', 'Carnaxide e Queijas').
paragem(1, 6, 593, -103777.02, -94637.67, 'Bom', 'Sem Abrigo', 'No', 'Vimeca', 300, 'Avenida dos Cavaleiros', 'Carnaxide e Queijas').
paragem(1, 7, 181, -103780.59, -96372.2, 'Bom', 'Aberto dos Lados', 'Yes', 'Vimeca', 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas').
paragem(1, 8, 180, -103842.39, -96260.96, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas').
paragem(1, 9, 594, -103879.91, -95751.23, 'Bom', 'Fechado dos Lados', 'No', 'Vimeca', 1116, 'Avenida Professor Dr. Reinaldo dos Santos', 'Carnaxide e Queijas').
paragem(1, 10, 185, -103922.82, -96235.62, 'Bom', 'Fechado dos Lados', 'Yes', 'SCoTTURB', 354, 'Rua Manuel Teixeira Gomes', 'Carnaxide e Queijas').
paragem(1, 11, 89, -103934.24, -96642.56, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1113, 'Avenida de Portugal', 'Carnaxide e Queijas').
paragem(1, 12, 107, -103972.32, -95981.88, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1113, 'Avenida de Portugal', 'Carnaxide e Queijas').
paragem(1, 13, 250, -104031.08, -96173.83, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1113, 'Avenida de Portugal', 'Carnaxide e Queijas').
paragem(1, 14, 261, -104032.88, -96536.98, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1113, 'Avenida de Portugal', 'Carnaxide e Queijas').
paragem(1, 15, 597, -104058.98, -95839.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1137, 'Rua Tenente-General Zeferino Sequeira', 'Carnaxide e Queijas').
paragem(1, 16, 953, -104075.89, -95771.82, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1116, 'Avenida Professor Dr. Reinaldo dos Santos', 'Carnaxide e Queijas').
paragem(1, 17, 609, -104226.49, -95797.22, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 327, 'Avenida do Forte', 'Carnaxide e Queijas').
paragem(1, 18, 242, -104235.94, -96573.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1279, 'Avenida Tomás Ribeiro', 'Carnaxide e Queijas').
paragem(1, 19, 255, -104240.6, -96543.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1279, 'Avenida Tomás Ribeiro', 'Carnaxide e Queijas').
paragem(1, 20, 604, -104256.82, -95173.34, 'Bom', 'Fechado dos Lados', 'No', 'Vimeca', 306, 'Rua dos Cravos de Abril', 'Carnaxide e Queijas').
paragem(1, 21, 628, -104278.8866659752, -94122.56603635015, 'Bom', 'Sem Abrigo', 'No', 'Vimeca', 1123, 'Rua da Quinta do Paizinho', 'Carnaxide e Queijas').
paragem(1, 22, 39, -104282.32, -95055.6, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 306, 'Rua dos Cravos de Abril', 'Carnaxide e Queijas').
paragem(1, 23, 50, -104287.85, -94105.37, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1123, 'Rua da Quinta do Paizinho', 'Carnaxide e Queijas').
```

Figura 23: Exemplos de predicados "paragem"para a Carreira 1

```
%% CARREIRA 1

adjacente(183, 791).
adjacente(791, 595).
adjacente(595, 182).
adjacente(182, 499).
adjacente(499, 593).
adjacente(593, 181).
adjacente(181, 180).
adjacente(180, 594).
adjacente(594, 185).
adjacente(185, 89).
adjacente(89, 107).
adjacente(107, 250).
adjacente(250, 261).
adjacente(261, 597).
adjacente(597, 953).
adjacente(953, 609).
```

Figura 24: Exemplos de predicados "adjacente"para a Carreira 1

A.3 Representação Visual das Carreiras

Figura 25: Carreiras Isoladas

Figura 26: Carreiras Interligadas