



UNIVERSIDADE DO MINHO

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO  
(3º ANO DE CURSO, 2º SEMESTRE)

---

## Exercício Individual

---

### RELATÓRIO DE DESENVOLVIMENTO

---

Mestrado Integrado em Engenharia Informática

*Realizado pelo aluno:*  
Filipa Alves dos Santos, a83631

5 de Junho de 2020

## Resumo

Devido à situação atual do ensino, que é praticado à distância por motivos de saúde pública, este trabalho foi solicitado aos alunos de modo a avaliar a componente individual da unidade curricular Sistemas de Representação de Conhecimento e Raciocínio, no ano letivo 19/20.

Este trabalho prático consiste na representação da base de conhecimento correspondente ao sistema de transportes do concelho de Oeiras, bem como a construção do sistema de recomendação, isto é, as queries (filtros) pedidas no enunciado.

O relatório começará por explicar o problema em si com uma breve introdução e, de seguida, será abordado o raciocínio tomado na execução deste projeto, na utilização dos dados fornecidos como também nos algoritmos feitos para filtrar a informação de acordo com os requisitos dados.

O objetivo principal deste trabalho é desenvolver a capacidade de interpretação do conhecimento como grafos e utilizar este raciocínio no sistema de recomendação através da linguagem de programação simbólica *PROLOG*.

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Preliminares</b>	<b>4</b>
<b>3</b>	<b>Descrição do Trabalho e Análise de Resultados</b>	<b>5</b>
3.1	Base de Conhecimento . . . . .	6
3.2	Query 1 . . . . .	9
3.3	Query 2 e 3 . . . . .	13
3.4	Query 4 . . . . .	15
3.5	Query 5 . . . . .	16
3.6	Query 6 . . . . .	18
3.7	Query 7 e 8 . . . . .	19
3.8	Query 9 . . . . .	21
<b>4</b>	<b>Conclusões e Sugestões</b>	<b>22</b>

## Lista de Figuras

1	Código do programa em Python . . . . .	6
2	Ficheiro PROLOG correspondente às paragens da carreira 1 . . . . .	6
3	Includes do conhecimento . . . . .	7
4	Código do segundo programa em Python . . . . .	8
5	Ficheiro PROLOG com todas as adjacências . . . . .	8
6	Predicado trajetoGeral . . . . .	9
7	Predicado trajeto . . . . .	9
8	Predicado auxTrajeto . . . . .	10
9	Predicado loopChkDestino . . . . .	10
10	Predicado loopChk . . . . .	10
11	Predicado verificaMesmaCarreira . . . . .	11
12	Predicado adjacentes . . . . .	11
13	Predicado adjacentes2 . . . . .	11
14	Predicado trajetoNaoInformado . . . . .	12
15	Algoritmo principal das queries 2 e 3 . . . . .	13
16	Predicados das queries 2 e 3 . . . . .	14
17	Predicados da query 4 . . . . .	15
18	Predicados da query 5 . . . . .	16
19	Algoritmo principal das queries 7 e 8 . . . . .	19
20	Predicados das queries 7 e 8 . . . . .	20
21	Predicados da query 9 . . . . .	21

## 1 Introdução

No enunciado fornecido pela equipa docente, o caso de estudo proposto são os dados do sistema de transportes do concelho de Oeiras. Estes dados sobre a rede de paragens de autocarros de Oeiras, que estão em ficheiros Excel no seu estado inicial, incluem informações sobre cada paragem, como as carreiras a que pertence, a respetiva operadora, o tipo de abrigo, entre outras. Serão a base de conhecimento do programa.

Como fase inicial, os dados foram processados para formato de predicado no código *PROLOG*. O processo de tal será descrito no início da próxima secção do relatório. Todos os dados de conhecimento imperfeito, isto é, paragens sem alguma informação, foram ignorados e descartados.

De seguida, estes dados, agora em forma de predicados "paragem" no ficheiro de código *PROLOG*, foram desenvolvidos vários algoritmos para conseguirmos percorrer a informação de maneira eficaz e rápida e obtendo os outputs desejados nas diversas queries pedidas no enunciado, como, por exemplo, determinar um percurso entre duas paragens excluindo um ou mais operadoras de transporte.

## 2 Preliminares

De modo a ter todos os recursos necessários para a execução deste trabalho, foi feita uma breve pesquisa sobre a matéria necessária para tal.

Sendo que as paragens formam uma rede isto é, um grafo, e todas as queries envolvem percursos entre paragens, facilmente se concluiu que iria ser necessários algoritmos de procura de árvores. Este assunto foi estudado nas aulas teórico-práticas da unidade curricular, através de fichas que tinham premissas semelhantes, bem como nas aulas teóricas.

Foram estudados vários tipos genéricos de algoritmos de procura, como o método de profundidade primeiro e o método da A estrela. Todos os tipos diferentes de procura têm as suas vantagens e a escolha de um tem de ser feita de acordo com o problema em questão.

Para este trabalho, foi pedido que utilizássemos dois tipos de **estratégias de pesquisa: não-informada e informada**. Uma estratégia informada consiste em percorrer as paragens utilizando a sua informação (como ,por exemplo, a carreira) para influenciar o rumo do percurso determinado. Já a não-informada, é uma estratégia sem qualquer dependência na informação sobre a paragem. A vantagem da primeira é que, por vezes, conseguimos os resultados mais rapidamente e é também muito útil para queries em que necessitamos de filtrar informação (por exemplo, quando queremos percorrer apenas paragens abrigadas). Por outro lado, a informada permite obter diferentes percursos para a mesma origem e destino sendo que pode haver variações na escolha do nodo seguinte no próprio algoritmo, permitindo assim, por exemplo, achar caminhos mais curtos.

No desenvolvimento desta resolução, foram utilizadas ambas as estratégias dependendo do problema em questão de modo a obter os resultados pretendidos o mais eficazmente possível.

### 3 Descrição do Trabalho e Análise de Resultados

Nesta primeira secção deste capítulo, [Base de Conhecimento](#), irá ser abordado o modo de como os dados foram tratados no povoamento da base de conhecimento. De seguida, irá haver uma secção para os requisitos do sistema de recomendação, explicando mais detalhadamente os pormenores dos algoritmos feitos:

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

### 3.1 Base de Conhecimento

O primeiro desafio relativamente à base de conhecimento foi como passar os dados fornecidos em *Excel* para predicados no programa de *PROLOG*.

Por sugestão da equipa docente, decidiu-se utilizar a biblioteca *Pandas* de *Python* para iterarmos nos ficheiros de formato *CSV*. Assim, depois de se exportar todas as páginas fornecidas no ficheiro "lista\_adjacencia\_paragens" como ficheiros individuais de extensão *CSV*, foi utilizado o seguinte código:

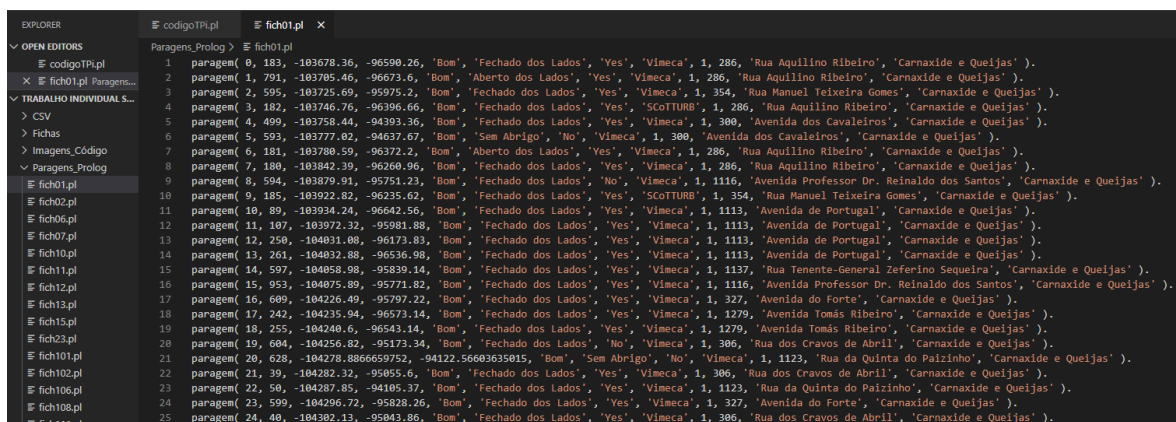
```
import pandas as pd
dados = pd.read_csv("lista_adjacencias_paragens_184.csv", encoding='utf-8').dropna()

saveFile = open('C:/Users/Filipa/Desktop/Paragens_Prolog/fich184.pl', 'w', encoding='utf-8')

for i in range(0,dados.shape[0]):
    saveFile.write('paragem( ')
    saveFile.write(str(i))
    saveFile.write(', ')
    for j in range(0,dados.shape[1]):
        if (j == 3 or j == 4 or j == 5 or j == 6 or j == 9 or j == 10) :
            saveFile.write("")
            saveFile.write(str(dados.iloc[i,j]))
            saveFile.write("")
        else : saveFile.write(str(dados.iloc[i,j]))
        if (j != 10) : saveFile.write(', ')
    saveFile.write(' ).')
    saveFile.write('\n')
saveFile.close()
```

Figura 1: Código do programa em Python

Correndo este código para os ficheiros correspondentes às 39 diferentes carreiras, obteve-se o resultado pretendido:



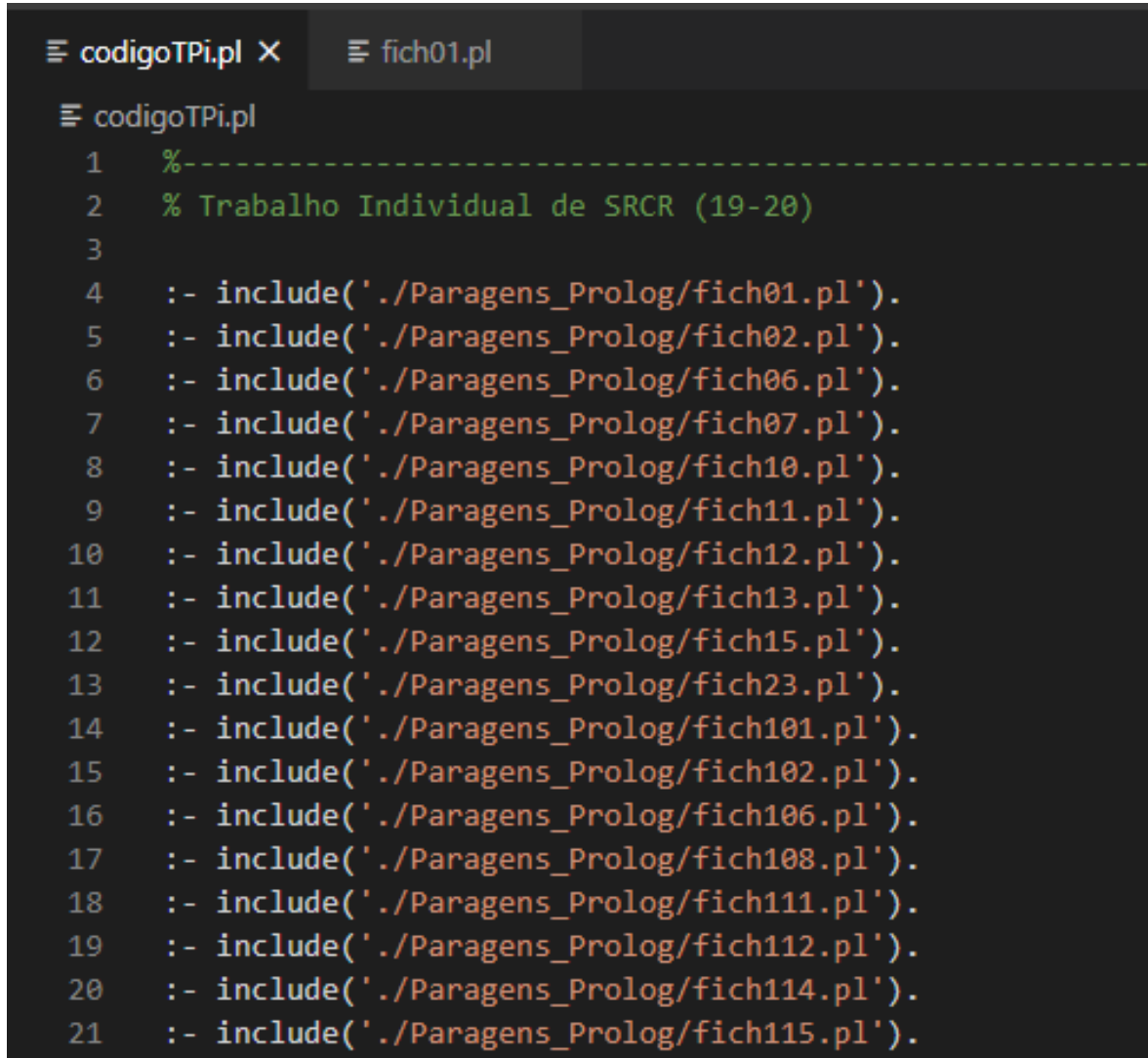
```
1 paragem( 0, 183, -103678.36, -96598.26, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas' ).
2 paragem( 1, 791, -103705.46, -96673.6, 'Bom', 'Aberto dos Lados', 'Yes', 'Vimeca', 1, 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas' ).
3 paragem( 2, 595, -103725.69, -95975.2, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 354, 'Rua Manuel Teixeira Gomes', 'Carnaxide e Queijas' ).
4 paragem( 3, 182, -103746.76, -96096.06, 'Bom', 'Fechado dos Lados', 'Yes', 'SCOTTURB', 1, 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas' ).
5 paragem( 4, 489, -103758.44, -94393.36, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 308, 'Avenida dos Cavaleiros', 'Carnaxide e Queijas' ).
6 paragem( 5, 593, -103777.02, -94637.67, 'Bom', 'Sem Abrigo', 'No', 'Vimeca', 1, 308, 'Avenida dos Cavaleiros', 'Carnaxide e Queijas' ).
7 paragem( 6, 181, -103780.59, -96372.2, 'Bom', 'Aberto dos Lados', 'Yes', 'Vimeca', 1, 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas' ).
8 paragem( 7, 180, -103842.39, -96260.96, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 286, 'Rua Aquilino Ribeiro', 'Carnaxide e Queijas' ).
9 paragem( 8, 594, -103879.91, -95751.23, 'Bom', 'Fechado dos Lados', 'No', 'Vimeca', 1, 1116, 'Avenida Professor Dr. Reinaldo dos Santos', 'Carnaxide e Queijas' ).
10 paragem( 9, 185, -103922.82, -96235.62, 'Bom', 'Fechado dos Lados', 'Yes', 'SCOTTURB', 1, 354, 'Rua Manuel Teixeira Gomes', 'Carnaxide e Queijas' ).
11 paragem( 10, 89, -103934.24, -96642.56, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1113, 'Avenida de Portugal', 'Carnaxide e Queijas' ).
12 paragem( 11, 187, -103972.32, -95981.88, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1113, 'Avenida de Portugal', 'Carnaxide e Queijas' ).
13 paragem( 12, 250, -104031.08, -96173.83, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1113, 'Avenida de Portugal', 'Carnaxide e Queijas' ).
14 paragem( 13, 261, -104032.88, -96536.98, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1113, 'Avenida de Portugal', 'Carnaxide e Queijas' ).
15 paragem( 14, 597, -104058.98, -95839.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1137, 'Rua Tenente-General Zeferino Sequeira', 'Carnaxide e Queijas' ).
16 paragem( 15, 953, -104075.89, -95771.82, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1116, 'Avenida Professor Dr. Reinaldo dos Santos', 'Carnaxide e Queijas' ).
17 paragem( 16, 609, -104226.49, -95797.22, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 327, 'Avenida do Forte', 'Carnaxide e Queijas' ).
18 paragem( 17, 242, -104235.94, -96573.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1279, 'Avenida Tomás Ribeiro', 'Carnaxide e Queijas' ).
19 paragem( 18, 255, -104240.6, -96543.14, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1279, 'Avenida Tomás Ribeiro', 'Carnaxide e Queijas' ).
20 paragem( 19, 604, -104256.82, -95173.34, 'Bom', 'Fechado dos Lados', 'No', 'Vimeca', 1, 306, 'Rua dos Cravos de Abril', 'Carnaxide e Queijas' ).
21 paragem( 20, 628, -104278.886659752, -94122.56603635015, 'Bom', 'Sem Abrigo', 'No', 'Vimeca', 1, 1123, 'Rua da Quinta do Paizinho', 'Carnaxide e Queijas' ).
22 paragem( 21, 39, -104282.32, -95055.6, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 306, 'Rua dos Cravos de Abril', 'Carnaxide e Queijas' ).
23 paragem( 22, 50, -104287.85, -94105.37, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 1123, 'Rua da Quinta do Paizinho', 'Carnaxide e Queijas' ).
24 paragem( 23, 599, -104296.72, -95828.26, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 327, 'Avenida do Forte', 'Carnaxide e Queijas' ).
25 paragem( 24, 40, -104302.13, -95843.86, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 1, 306, 'Rua dos Cravos de Abril', 'Carnaxide e Queijas' ).
```

Figura 2: Ficheiro PROLOG correspondente às paragens da carreira 1

Para além da informação de cada paragem, o predicado vai também guardar a posição em que está na carreira em questão, no 1º argumento. Isto será útil para algoritmos explicados



posteriormente. De modo a conseguir utilizar este predicado, fez-se `include` de todos estes ficheiros no principal, que contém o restante código das queries:



```
≡ codigoTPi.pl X  ≡ fich01.pl
≡ codigoTPi.pl
1  %-----
2  % Trabalho Individual de SRCR (19-20)
3
4  :- include('./Paragens_Prolog/fich01.pl').
5  :- include('./Paragens_Prolog/fich02.pl').
6  :- include('./Paragens_Prolog/fich06.pl').
7  :- include('./Paragens_Prolog/fich07.pl').
8  :- include('./Paragens_Prolog/fich10.pl').
9  :- include('./Paragens_Prolog/fich11.pl').
10 :- include('./Paragens_Prolog/fich12.pl').
11 :- include('./Paragens_Prolog/fich13.pl').
12 :- include('./Paragens_Prolog/fich15.pl').
13 :- include('./Paragens_Prolog/fich23.pl').
14 :- include('./Paragens_Prolog/fich101.pl').
15 :- include('./Paragens_Prolog/fich102.pl').
16 :- include('./Paragens_Prolog/fich106.pl').
17 :- include('./Paragens_Prolog/fich108.pl').
18 :- include('./Paragens_Prolog/fich111.pl').
19 :- include('./Paragens_Prolog/fich112.pl').
20 :- include('./Paragens_Prolog/fich114.pl').
21 :- include('./Paragens_Prolog/fich115.pl').
```

Figura 3: Includes do conhecimento

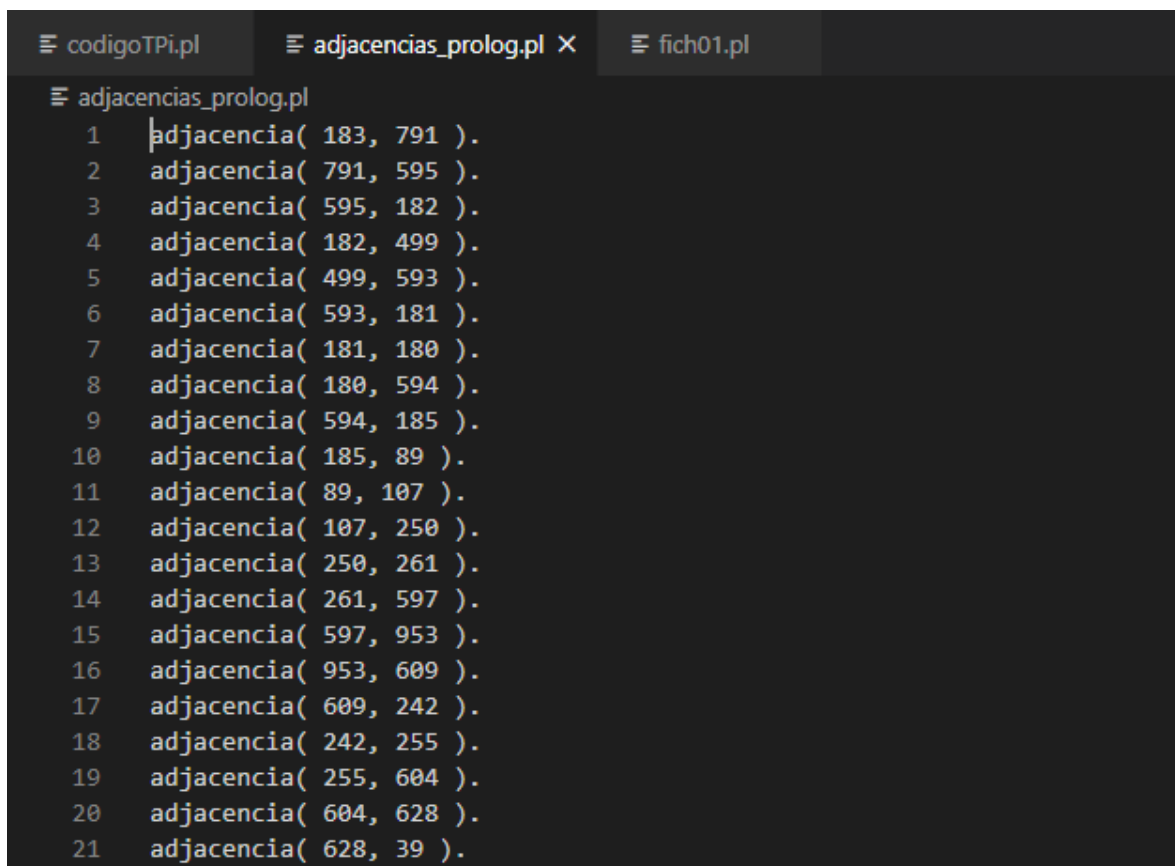
Além disto, com o mesmo ficheiro "`lista_adjacencia_paragens`", também se criou uma lista de adjacências para facilitar a query do caminho mais curto. Para tal, foi necessário outro programa *Python* (figura 4) que cria todos os pares adjacentes, retirando todos os repetidos e, tal como no primeiro programa, todas as paragens que têm conhecimento imperfeito, através da função `dropna()`.

O ficheiro resultante terá todos os pares de paragens adjacentes (não repetidos), em forma do predicado `adjacencia`.

```
import pandas as pd
import os
saveFile = open('C:/Users/Filipa/Desktop/adjacencias_prolog.pl','w', encoding='utf-8')

adjacenciasVisitadas=[]
for ficheiro in os.listdir('C:/Users/Filipa/Desktop/Current/Trabalho Individual SRCR/CSV/'):
    dados = pd.read_csv('C:/Users/Filipa/Desktop/Current/Trabalho Individual SRCR/CSV/' + ficheiro, encoding='utf-8').dropna()
    for i in range(0,dados.shape[0] - 1):
        if(not((dados.iloc[i,0]),(dados.iloc[i+1,0]))in(adjacenciasVisitadas)):
            adjacenciasVisitadas.append(((dados.iloc[i,0]),(dados.iloc[i+1,0])))
            saveFile.write('adjacencia( ')
            saveFile.write(str(dados.iloc[i,0]))
            saveFile.write(', ')
            saveFile.write(str(dados.iloc[i+1,0]))
            saveFile.write(' ).\n')
            saveFile.write('\n')
saveFile.close()
```

Figura 4: Código do segundo programa em Python



```
≡ adjacencias_prolog.pl
1  adjacencia( 183, 791 ).
2  adjacencia( 791, 595 ).
3  adjacencia( 595, 182 ).
4  adjacencia( 182, 499 ).
5  adjacencia( 499, 593 ).
6  adjacencia( 593, 181 ).
7  adjacencia( 181, 180 ).
8  adjacencia( 180, 594 ).
9  adjacencia( 594, 185 ).
10 adjacencia( 185, 89 ).
11 adjacencia( 89, 107 ).
12 adjacencia( 107, 250 ).
13 adjacencia( 250, 261 ).
14 adjacencia( 261, 597 ).
15 adjacencia( 597, 953 ).
16 adjacencia( 953, 609 ).
17 adjacencia( 609, 242 ).
18 adjacencia( 242, 255 ).
19 adjacencia( 255, 604 ).
20 adjacencia( 604, 628 ).
21 adjacencia( 628, 39 ).
```

Figura 5: Ficheiro PROLOG com todas as adjacências

### 3.2 Query 1

Neste primeiro ponto, foi pedido para determinar um caminho entre 2 pontos, isto é, 2 paragens. Logo, o predicado principal `trajetoGeral` recebe o gid (identificador único) da origem e do destino e devolve um caminho possível:

```
% Algoritmo final query1
trajetoGeral(Origem, Destino, [Origem|Final]) :- trajeto(Origem, Destino, [Origem], [], Final).
```

Figura 6: Predicado `trajetoGeral`

Já o predicado `trajeto` vai chamar o `auxTrajeto` para determinar a próxima paragem no percurso a tomar. No caso de o `auxTrajeto` ser bem sucedido, o `trajeto` adiciona-a à lista de paragens visitadas bem como à lista do caminho atual e avança o processo. Caso contrário, este predicado auxiliar vai devolver 0 e o algoritmo irá entrar na última definição do `trajeto` em que se faz **backtracking**. Isto é, o caminho poderá ter entrado num ciclo em que uma paragem não tem adjacentes ainda não visitadas e então, irá voltar atrás para tentar outro caminho em que tal não aconteça. O caso de paragem deste algoritmo é quando a paragem atual coincide com o destino, isto é, já temos o nosso caminho final determinado.

```
% Algoritmo principal para estabelecer o trajeto
trajeto(Destino, Destino, _, Final, RFinal):- reverse(Final,RFinal).

trajeto(Paragem, Destino, Visitadas, Caminho, Final):-
    auxTrajeto(Paragem, Destino, Visitadas, ProxParagem),
    ProxParagem \= 0 ,!,
    trajeto(ProxParagem, Destino, [ProxParagem|Visitadas], [ProxParagem|Caminho], Final).

trajeto(Paragem, Destino, Visitadas, [Paragem,P2|RestoPercurso], Final):- % Backtracking
    trajeto(P2, Destino, Visitadas, [P2|RestoPercurso], Final).
```

Figura 7: Predicado `trajeto`

No `auxTrajeto` temos duas definições distintas. De modo a que o algoritmo seja o mais rápido possível, primeiro verifica se a paragem atual já está na mesma carreira que o destino. Se for o caso, o algoritmo continuará a ir por esse caminho através do predicado `adjacentes2`. Caso contrário, executa a segunda definição do predicado que começa por determinar todas as paragens adjacentes à atual com o predicado `adjacentes`. De seguida, o `loopChkdestino` tenta escolher algum adjacente que tenha uma carreira em comum com o destino, com o mesmo objetivo de tentar encurtar o tempo de execução. Caso não haja nenhum que corresponda a este padrão, o `loopChk` vai escolher, se conseguir, o primeiro adjacente que ainda não tenha sido visitado.

```
% Predicado auxiliar, que faz o ciclo para verificar todas as paragens adjacentes
auxTrajeto(Paragem, Destino, _, ProxParagem):- % verificaMesmaCarreira devolve carreira em comum (do destino)
    verificarMesmaCarreira(Paragem, Destino, Car),
    adjacentes2(Paragem, Car, ProxParagem).

auxTrajeto(Paragem, Destino, Visitadas, ProxParagem):- % verificaMesmaCarreira devolve false
    adjacentes(Paragem, Adjacentes),
    loopChkDestino(Adjacentes, Destino, Visitadas, Prox),
    ((Prox =\= 0,!, ProxParagem is Prox) ; (loopChk(Adjacentes, Visitadas, ProxParagem))).
```

Figura 8: Predicado auxTrajeto

Tal como mencionado anteriormente, o predicado `loopChkDestino` vai percorrer a lista das paragens adjacentes e devolverá uma paragem se esta estiver na carreira do destino e se ainda não tiver sido visitada. Além disso, como se considerou carreiras unidireccionais, o destino tem de estar depois da paragem em questão, isto é, a sua ordem tem de ser superior. Caso não tenha sucesso, devolverá a flag 0.

```
% Ciclo para verificar Visitadas + se há algum adjacente com a carreira do atual
loopChkDestino([], _, _, 0). % Todos os adjacentes foram visitados + não há nenhum adjacente na carreira do destino (devolve flag 0)

loopChkDestino([H|_], Destino, Visitadas, H):- % Head é uma paragem não visitada
    \+ memberchk(H, Visitadas),
    paragem(OrdH, H, _, _, Car, _, _),
    paragem(OrdD, Destino, _, _, Car, _, _),
    OrdD > OrdH.

loopChkDestino([_|T], Destino, Visitadas, ProxParagem):- % Head já foi visitada e/ou não estava na carreira do destino
    loopChkDestino(T, Destino, Visitadas, ProxParagem).
```

Figura 9: Predicado loopChkDestino

O `loopChk` é semelhante ao `loopChkDestino`, mas só verifica se a paragem já foi visitada ou não. Devolverá a primeira adjacente que encontrar que cumpra tal regra ou, caso não encontre, 0.

```
% Ciclo para verificar Visitadas
loopChk([], _, 0). % Todos os adjacentes foram visitados (devolve flag 0)

loopChk([H|_], Visitadas, H):- % Head não visitada ainda
    \+ memberchk(H, Visitadas).

loopChk([_|T], Visitadas, ProxParagem):- % Head já foi visitada
    loopChk(T, Visitadas, ProxParagem).
```

Figura 10: Predicado loopChk

O predicado `verificaMesmaCarreira` vai verificar se a paragem atual já se encontra na mesma carreira que o destino:

```
% Verifica se a Paragem (atual) e o Destino têm a mesma Carreira
✓ verificarMesmaCarreira(Paragem, Destino, Car):-
    paragem(OrdP, Paragem, _, _, _, _, _, Car, _, _, _),
    paragem(OrdD, Destino, _, _, _, _, _, Car, _, _, _),
    OrdD > OrdP.
```

Figura 11: Predicado verificaMesmaCarreira

O predicado `adjacentes` irá devolver uma lista de todas as paragens adjacentes (os seus gids) a uma certa paragem. Serão consideradas adjacentes se tiverem imediatamente a seguir à paragem fornecida e partilharem uma carreira:

```
% Determina todas as paragens adjacentes
adjacentes(Paragem, Adjacentes):-
    findall(ProxParagem, ( paragem(Ord, Paragem, _, _, _, _, _, Car, _, _, _),
                          OrdInc is Ord+1,
                          paragem(OrdInc, ProxParagem, _, _, _, _, _, Car, _, _, _),
                          ), Adjacentes).
```

Figura 12: Predicado adjacentes

Semelhante ao `adjacentes`, o predicado `adjacentes2` irá apenas selecionar o adjacente da paragem dada, na carreira também fornecida como argumento:

```
% "adjacentes" mais simplificado, para quando já sabemos que estamos na carreira do Destino
adjacentes2(Paragem, Car, ProxParagem):-
    paragem(Ord, Paragem, _, _, _, _, _, Car, _, _, _),
    OrdInc is Ord+1,
    paragem(OrdInc, ProxParagem, _, _, _, _, _, Car, _, _, _).
```

Figura 13: Predicado adjacentes2

Todo este algoritmo descrito através destes predicados utilizou uma estratégia **informada**, utilizando dados como a ordem, a carreira, consultando uma lista de paragens visitadas, etc. Já o algoritmo seguinte, apesar de não muito utilizado no resto do trabalho, seria uma versão **não-informada**. Este predicado `trajetoNaoInformado` simplesmente irá construir o caminho profundidade-primeiro, sem qualquer otimização.

```
adjacente(Paragem1, Paragem2) :-  
    paragem(Ord, Paragem1, _, _, _, _, _, Car, _, _, _),  
    OrdInc is Ord+1,  
    paragem(OrdInc, Paragem2, _, _, _, _, _, Car, _, _, _).  
  
trajetoNaoInformado(Origem, Destino, [Origem, Destino]) :-  
    adjacente(Origem, Destino).  
  
trajetoNaoInformado(Origem, Destino, [Origem|T]) :-  
    adjacente(Origem, Temp),  
    trajetoNaoInformado(Temp, Destino, T).
```

Figura 14: Predicado trajetoNaoInformado

### 3.3 Query 2 e 3

As queries 2 e 3 pediam para selecionar apenas algumas das operadoras de transporte para um determinado percurso e excluir um ou mais operadores de transporte para o percurso, respetivamente. Basicamente todo o seu raciocínio é equivalente a uma versão simplificada do algoritmo da query 1. A mudança mais significativa é que, neste caso, como já queremos limitar o caminho de acordo com as operadoras, não verificamos quaisquer carreiras com a intenção de influenciar o caminho.

```
% Ciclo para verificar Visitadas + requirimento da query especifica
loopChk2(_,[],_,_, 0). % Todos os adjacentes foram visitados (devolve flag 0)

loopChk2(2, [H|_], Operadoras, Visitadas, H):- % Head não visitada ainda + operadora pertence às selecionadas
    \+ memberchk(H, Visitadas),
    paragem(_,H,_,_,_,_,OpH,_,_,_,_),
    memberchk(OpH, Operadoras).

loopChk2(3, [H|_], Operadoras, Visitadas, H):- % Head não visitada ainda + operadora não pertence às excluídas
    \+ memberchk(H, Visitadas),
    paragem(_,H,_,_,_,_,OpH,_,_,_,_),
    \+ memberchk(OpH, Operadoras).

loopChk2(Query, [_|T], Operadoras, Visitadas, ProxParagem):- % Head já foi visitada
    loopChk2(Query, T, Operadoras, Visitadas, ProxParagem).

% Predicado auxiliar, que faz o ciclo para verificar todas as paragens adjacentes
auxquery(Query, Paragem, Operadoras, Visitadas, ProxParagem):-
    adjacentes(Paragem, Adjacentes),
    loopChk2(Query, Adjacentes, Operadoras, Visitadas, ProxParagem).

% Algoritmo principal para estabelecer o trajeto
query(_, Destino, Destino, _,_, Final, RFinal):- reverse(Final,RFinal).

query(Query, Paragem, Destino, Operadoras, Visitadas, Caminho, Final):-
    auxquery(Query, Paragem, Operadoras, Visitadas, ProxParagem),
    ProxParagem =\= 0 ,!,
    query(Query, ProxParagem, Destino, Operadoras, [ProxParagem|Visitadas], [ProxParagem|Caminho], Final).

query(Query, Paragem, Destino, Operadoras, Visitadas, [Paragem,P2|RestoPercurso], Final):- %Backtracking
    query(Query, P2, Destino, Operadoras, Visitadas, [P2|RestoPercurso], Final).
```

Figura 15: Algoritmo principal das queries 2 e 3

No caso da query 2 (predicado `algumasOpGeral`), irá fazer um `memberchk` adicional na `loopChk2` para se certificar de que a operadora da paragem adjacente escolhida terá de pertencer à lista de operadoras fornecida. Na query 3 (predicado `excluirOpGeral`), verifica-se o oposto, isto é, se a operadora da paragem adjacente escolhida não pertence à mesma lista.

```
algumasOpGeral(Origem, Destino, Operadoras, [Origem|Final]) :-  
    paragem(_,Origem,_,_,_,_,OpH,_,_,_,_),  
    \+ memberchk(OpH, Operadoras),  
    query(2, Origem, Destino, Operadoras, [Origem], [], Final).  
  
excluirOpGeral(Origem, Destino, Operadoras, [Origem|Final]) :-  
    paragem(_,Origem,_,_,_,_,OpH,_,_,_,_),  
    memberchk(OpH, Operadoras),  
    query(3, Origem, Destino, Operadoras, [Origem], [], Final).
```

Figura 16: Predicados das queries 2 e 3



### 3.4 Query 4

Esta query pedia para identificar quais as paragens com o maior número de carreiras num determinado percurso. Primeiro, fez-se proveito do predicado `trajetoGeral` para determinar um caminho entre a origem e o destino dados. De seguida, é utilizado o predicado `maisCarreiras` que, dado um caminho, devolve uma lista de pares em que cada par corresponde ao gid das paragens do caminho e o respetivo número de carreiras a que pertence cada uma. Por fim, o `xCarreiras` é utilizado para determinar esse tal número para cada paragem.

No predicado principal, `maisCarreiras`, depois de determinada a lista dos pares, são utilizados os predicados predefinidos `keysort` e `reverse` para os resultados serem apresentados de modo decrescente de número de carreiras.

```
% Acha o número de carreiras para uma paragem (gid)
xCarreiras(H,NrCarreirasHead) :-
    findall(Car,(paragem(_, H, _, _, _, _, _, Car, _, _, _)),CarreirasTodas),
    length(CarreirasTodas, NrCarreirasHead).

% Algoritmo principal, itera o percurso e guarda paragem e o número de carreiras correspondente num par
maisCarreiras([H], [(NrCarreirasHead-H)]) :- xCarreiras(H,NrCarreirasHead).

maisCarreiras([H|T], [(NrCarreirasHead-H)|TC]) :-
    xCarreiras(H,NrCarreirasHead),
    maisCarreiras(T,TC).

% Algoritmo final query4
maisCarreirasGeral(Origem, Destino, MaxCarreiras) :-
    trajetoGeral(Origem, Destino, Caminho),
    maisCarreiras(Caminho, NrCarreiras),
    keysort(NrCarreiras, MaxCarreirasDec),
    reverse(MaxCarreirasDec, MaxCarreiras).
```

Figura 17: Predicados da query 4

### 3.5 Query 5

Esta query, que pede o caminho mais curto entre dois pontos, foi a única em que foi necessário utilizar o método não-informado de pesquisa.

Utilizamos o predicado `trajetoNaoInformadoLim`, que é igual ao `trajetoNaoInformado` mencionado na seção da query 1. A única diferença é que este tem um argumento extra correspondente ao máximo número de iterações (que corresponderá ao número de paragens) desejada. Isto é, vai tentar fazer o caminho dentro do número fornecido.

```
trajetoNaoInformadoLim(_, _, Maximo, Maximo, _) :- !,fail.

trajetoNaoInformadoLim(Origem, Destino, _, _, [Origem, Destino]) :-
    adjacencia(Origem, Destino).

trajetoNaoInformadoLim(Origem, Destino, Iteracao, Maximo, [Origem|T]) :-
    IteracaoNova is Iteracao + 1,
    adjacencia(Origem, Temp),
    trajetoNaoInformadoLim(Temp, Destino, IteracaoNova, Maximo, T).

maisCurtoGeral(Origem, Destino, Caminho) :-
    trajetoGeral(Origem, Destino, CaminhoMax), % verifica se caminho existe
    length(CaminhoMax, L),
    maisCurto(Origem, Destino, L, Caminho).

maisCurto(Origem, Destino, L, Caminho) :-
    desce(Origem, Destino, L, NovoL),
    sobe(Origem, Destino, NovoL, Caminho).

desce(Origem, Destino, L, Caminho) :-
    Lnovo is L-5,
    trajetoNaoInformadoLim(Origem, Destino, 1, Lnovo, _),
    desce(Origem, Destino, Lnovo, Caminho).

desce(_, _, L, L2) :- L2 is L-5.

sobe(Origem, Destino, L, Caminho) :-
    trajetoNaoInformadoLim(Origem, Destino, 1, L, Caminho).

sobe(Origem, Destino, L, Caminho) :-
    NovoL is L+1,
    sobe(Origem, Destino, NovoL, Caminho).
```

Figura 18: Predicados da query 5

O predicado principal, `maisCurtoGeral`, começa por verificar a existência de um caminho com a pesquisa informada `trajetoGeral`. Caso exista, o tamanho do caminho será



determinado e estabelecido como o máximo.

De seguida, o **maisCurto** vai começar por descer este valor. Vai utilizar o predicado **desce** que vai repetir o processo de achar um trajeto (com a pesquisa não-informada **trajetoNaoInformadoLim**) para um número de paragens cada vez menor, até ser impossível. Quando este ponto for atingido, outro predicado, **sobe**, é utilizado para subir 1 a 1 esse valor até que consiga achar um trajeto possível. Esse percurso será o mais curto possível e é devolvido no predicado principal.



### 3.6 Query 6

Infelizmente, devido a limitações de tempo, esta query não foi resolvida.

### 3.7 Query 7 e 8

Tal como as queries 2 e 3, estas também têm um raciocínio idêntico à query 1.

Na query 7 foi pedido para escolher o percurso que passe apenas por abrigos com publicidade e na 8 para escolher o percurso que passe apenas por paragens abrigadas.

```
% Ciclo para verificar Visitadas + verificar publicidade
loopChk3(_, [], _, 0). % Todos os adjacentes foram visitados (devolve flag 0)

loopChk3(7, [H|_], Visitadas, H):- % Head não visitada ainda + tem publicidade
    \+ memberchk(H, Visitadas),
    paragem(_, H, _, _, Pub, _, _, _),
    Pub == 'Yes'.

loopChk3(8, [H|_], Visitadas, H):- % Head não visitada ainda + paragem é abrigada
    \+ memberchk(H, Visitadas),
    paragem(_, H, _, _, Tipo, _, _, _),
    (Tipo == 'Aberto dos Lados'; Tipo == 'Fechado dos Lados').

loopChk3(Query, [_|T], Visitadas, ProxParagem):- % Head já foi visitada
    loopChk3(Query, T, Visitadas, ProxParagem).

% Predicado auxiliar, que faz o ciclo para verificar todas as paragens adjacentes
auxquery2(Query, Paragem, Visitadas, ProxParagem):-
    adjacentes(Paragem, Adjacentes),
    loopChk3(Query, Adjacentes, Visitadas, ProxParagem).

% Algoritmo principal para estabelecer o trajeto
query2(_, Destino, Destino, _, Final, RFinal):- reverse(Final, RFinal).

query2(Query, Paragem, Destino, Visitadas, Caminho, Final):-
    auxquery2(Query, Paragem, Visitadas, ProxParagem),
    ProxParagem \= 0, !,
    query2(Query, ProxParagem, Destino, [ProxParagem|Visitadas], [ProxParagem|Caminho], Final).

query2(Query, Paragem, Destino, Visitadas, [Paragem, P2|RestoPercurso], Final):- %Backtracking
    query2(Query, P2, Destino, Visitadas, [P2|RestoPercurso], Final).
```

Figura 19: Algoritmo principal das queries 7 e 8

Assim, na query 7 (predicado `publicidadeGeral`) é verificado para todas as paragens escolhidas se existe publicidade e na 8 (predicado `abrigadasGeral`) é verificado para todas as paragens escolhidas se são abrigadas. Estas verificações são logo feitas na origem e no predicado `loopChk3`, quando se escolhe a próxima paragem para onde avançar.

```
% Algoritmo final query7
publicidadeGeral(Origem, Destino, [Origem|Final]) :-
    paragem(_,Origem,_,_,_,Pub,_,_,_,_,_),
    Pub == 'Yes',
    query2(7, Origem, Destino, [Origem], [], Final).

% Algoritmo final query8
abrigadasGeral(Origem, Destino, [Origem|Final]) :-
    paragem(_,Origem,_,_,_,Tipo,_,_,_,_,_),
    (Tipo == 'Aberto dos Lados'; Tipo == 'Fechado dos Lados'),
    query2(8, Origem, Destino, [Origem], [], Final).
```

Figura 20: Predicados das queries 7 e 8

### 3.8 Query 9

Nesta query, era pedido que se pudesse escolher um ou mais pontos intermédios por onde o percurso deverá passar. O raciocínio utilizado foi bastante simples: utilizar o predicado `trajetoGeral` para determinar os vários caminhos entre os pontos consecutivos e concatenar estes caminhos.

Assim, o predicado `intermediosGeral` começa por fazer o caminho entre a origem o primeiro ponto fornecido. De seguida, o predicado `intermedios` faz o mesmo raciocínio para todos os caminhos intermédios, com uso do `concatListas` que junta dois caminhos num só.

```
% Junta duas listas de modo a não repetir o ponto em comum
concatListas(Caminho1,[_|T2],Concatado) :- append(Caminho1, T2, Concatado).

% Algoritmo principal que faz os percursos intermédios e junta-os
intermedios(Destino, [H,[]|[]], Atual, Final2) :-
    trajetoGeral(H, Destino, CaminhoL),
    concatListas(Atual, CaminhoL, Final2).

intermedios(Destino, [H1,H2|R], Atual, Final2) :-
    trajetoGeral(H1, H2, CaminhoX),
    concatListas(Atual, CaminhoX, FinalTemp),
    intermedios(Destino, [H2,R], FinalTemp, Final2).

% Algoritmo final query9
intermediosGeral(Origem, Destino, [H|R], Final) :-
    trajetoGeral(Origem, H, CaminhoP),
    intermedios(Destino, [H|R], CaminhoP, Final).
```

Figura 21: Predicados da query 9

## 4 Conclusões e Sugestões

Para este projeto foi-me dada a oportunidade de pensar em estratégias de pesquisas em grafos num novo paradigma, que é a programação em lógica. Ajudou-me a cimentar conhecimento sobre pesquisas informadas e desinformadas e aprender sobre o mecanismo de inferência do SICStus e do SWI-Prolog. No entanto, gostaria de no futuro rever os meus predicados uma vez que não me encontro satisfeita de ter deixado uma alínea incompleta, relativamente ao cálculo do caminho mais rápido, em virtude do espaço amostral em relação à memória disponível do Prolog ser muito alto.