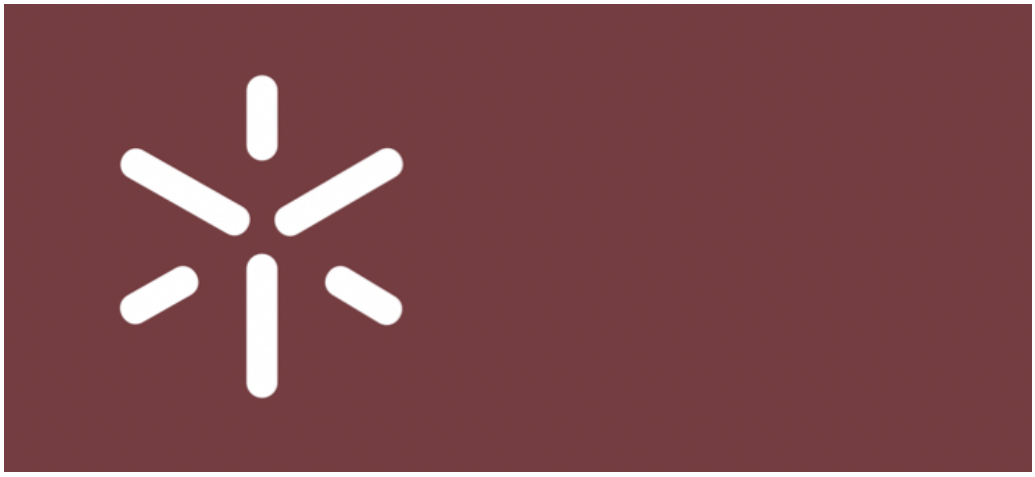

UNIVERSIDADE DO MINHO

Sistemas de Representação de Conhecimento e Raciocínio

TP2 - Sistema de recomendação para Transportes públicos

Rafael Lourenço
(A86266)

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

No âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, foi proposto desenvolver um sistema recomendação de transportes públicos. Este relatório visa detalhar uma implementação em *prolog* de uma possível solução para este problema. Neste projeto é apenas representado conhecimento perfeito, visto não ser necessário a representação do conhecimento imperfeito.
A criação da base de conhecimento também foi implementada em *prolog*, porem devido a necessidade de utilização de uma biblioteca, não foi possível usar unicamente o *Sictus* e foi usado *SWI*.
Para alem disso, são efectuadas técnicas de pesquisa em grafos, que neste caso representa os caminhos possíveis feito por um transportador, por exemplo um autocarro.
Os principais objetivos deste trabalho prático consistem em calcular o percurso entre duas paragem, com ou sem restrições.

Capítulo 2

Descrição do Trabalho

Neste capítulo,irei dividi-lo em várias partes para dar mais ênfase a certas fases, mais importantes no projeto. Em primeiro lugar, comecei por desenvolver predicados capazes de abrir e ler um ficheiro csv e para isso usei alguns predicados da biblioteca *library(csv)*, apenas possível usar no SWI.
Estes predicados são necessários para a criação da base de conhecimento,sendo apresentados na secção da criação da base de conhecimento.
Após ter criado a Base de conhecimento, foi necessário remover algumas incongruências que esta apresentava, como por exemplo, a existência de paragens que no grafo não existiam.Para remover essas irregularidades usei uma função em java(apresentada nos anexos), que me permitiu,de uma forma rápida, resolver esse problema.
Também usei algumas expressões regulares para remover conhecimento que era imperfeito e que poderia causar alguns problema com o sicstus.Um exemplo disso é este:

```
"paragem\(705.*\n
```

Este serviu para remover a paragem 705, pois essa não existe no grafo. Após este processo de limpeza de dados, temos a base de conhecimento perfeito praticamente criada.
De seguida, foi feito o algoritmo para pesquisar um caminho no grafo, que poderá ser visto implementado abaixo na secção da pesquisa no grafo.Foram criados 2 algoritmos de pesquisa baseados no algoritmo A$*$ apresentado nas aulas.
O primeiro passo do algoritmo é obter as carreira da origem e do destino.De seguida, é calculado se existem carreiras em comum, e caso existam basta viajar nessa carreira até ao destino. É aqui que este algoritmo difere do outro, uma vez que, no outro algoritmo , a cada viagem que realiza,é calculado novamente se existem carreiras em comum e caso existam ele move-se para aquela que apresenta menor custo.Esse processo é realizado por uma função de heurística, que é baseada nas coordenadas GPS de cada paragem,sendo assim escolhida aquela que apresenta menor distância.
Por outro lado, caso não existam carreiras em **comum** é executado um predicado que calcula todas as paragens adjacentes para todas as carreiras possíveis.Após obter todas,as de heurística para saber em que sentido é melhor avançar na carreira atual.
É de notar que é guardada numa lista todas as paragens por onde passamos para evitar ciclos infinitos no grafo.
Além destes algoritmos, também foi necessário criar predicados que removem possíveis viagens do grafo. Por Exemplo, para selecionar apenas alguma operadoras é necessário remover todos as viagens que não pertencem a essas operadoras.Outros exemplos são mostrados abaixo. Por último temos o calculo da duração da viagem,que é feito através de predicados auxiliares que nos indicam a hora atual,horas de ponta, horário de funcionamento, etc. As horas de ponta têm um coeficiente associado,e este representa no máximo um

atraso de 15 minutos entre paragens, que é obtido exatamente no meio da mesma.Por exemplo, se as horas de ponta forem entre as 13h e as 14h, temos atraso máximo às 13h e 30 minutos.

As Horas são representadas com dois inteiros,uma para as horas e outro para os minutos.É possível aterra a hora atual através de um predicado chamado *mudaHora*. Assim conclui a descrição do projeto. De seguida será apresentado por secções os predicados que realizam este processo.

2.1 Evolução de conhecimento

Estes predicados apresentados abaixo servem para evoluir a base do conhecimento. Neste trabalho é usado para a mudança da hora atual do sistema e do horário do funcionamento, que está fixado no enunciado, sendo então permitido viajar desde as 6 horas até 24h.O processo de mudar a hora de funcionamento não foi implementado, visto ser um objetivo secundário e irrevelante para esta fase.

```
evolucao( Termo ) :-
        findall( Invariante,+Termo::Invariante,Lista ),
        insercao( Termo ),
        teste( Lista ).
teste( [] ).
teste( [R|LR] ) :-
    R,
    teste( LR ).
insercao( Termo ) :-
    assert( Termo ).
insercao( Termo ) :-
    retract( Termo ), !,fail.

remocao( Termo ) :-
    retract( Termo ).
remocao( Termo ) :-
    assert( Termo ),!,fail.
```

Figura 2.1: Meta-predicado para a evolução/involução da base de conhecimento.

```
nao( Questao ) :-
    Questao, !, fail.
nao( Questao ).
```

Figura 2.2: Predicado que corresponde a extensao do meta-predicado não.

2.2 Predicados Auxiliares na criação da base de conhecimento

Nesta sub-secção procedi a implementação das funções auxiliares para a criação da base de conhecimento e do grafo, como é descrito de seguida.

Este predicado é usado na criação da base de conhecimento, para colocar uma lista das carreiras, nas paragens.

```
atomic_list_concat_(L, Sep, Atom) :-
        ( atom(Sep), ground(L), is_list(L) )
    ->  list_atom(L, Sep, Atom)
    ;   ( atom(Sep), atom(Atom) )
    ->  atom_list(Atom, Sep, L)
    ;   instantiation_error(atomic_list_concat_(L, Sep, Atom))
    .
```

Figura 2.3: Predicado que faz o split de uma string .

```
criarBC(Name_Exel,DirDestino,criada_Com_Sucesso) :-
    csv_read_file(Name_Exel, Data,[functor(paragem),arity(11),separator(0';)]), %'
                open(DirDestino,write,OS),
                mapBC(OS,Data,sucesso),
                close(OS).
    mapBC(OS,[],sucesso).
    mapBC(OS,[Y1|YS],R):- processaPar(Y1,Y),
                          writeq(OS,Y),
                          write(OS,"."),
                          nl(OS),
                          mapBC(OS,YS,R).
```

Figura 2.4: Criação da base de conhecimento.

```
processaPar(
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
            Operadora1,Carr1,CodRua1,NomeRua1,Freguesia1),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
            Operadora1,[Carr1],CodRua1,NomeRua1,Freguesia1)):-integer(Carr1).
processaPar(
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
            Operadora1,Carr1,CodRua1,NomeRua1,Freguesia1),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
            Operadora1,Carr2,CodRua1,NomeRua1,Freguesia1)):-
            atomic_list_concat_(L,',',Carr1),alltoInt(L,Carr2) .
processaPar(X,X).
```

Figura 2.5: Predicado auxiliar que colocar a lista das carreiras em inteiros.

```
alltoInt([],[]).
alltoInt([X|XS],[Y|YS]):-
        atom_number(X,Y),alltoInt(XS,YS).
```

Figura 2.6: Predicado que transforma uma lista de strings em inteiros.

```
criarGrafo(Name_Exel,DirDestino,criado_Com_Sucesso) :-
    csv_read_file(Name_Exel,Data,[functor(paragem),arity(11),separator(0';)]), %'
                    open(DirDestino,write,OS),
                    write(OS,"grafo("),
                    todasParagens(Data,Res),length(Res,P),
                    write(OS,Res),nl(OS),write(OS,","),
                    agrupaParagem(Data,Pares),length(Pares,C),
                    writeq(OS,Pares),write(OS,").",
                    nl(OS),close(OS),
                    print("Grafo com"), print(C),
                    print("Arcos(caminhos) e "), print(P),
                    print(" vertices(paragens) !").
```

Figura 2.7: Predicado que cria o grafo.

```
todasParagens([],R).
todasParagens([paragem(X1,_,_,_,_,_,_,_,_,_,_)|YS],XS):-
        memberchk(X1,XS),todasParagens(YS,XS).
todasParagens([paragem(X1,_,_,_,_,_,_,_,_,_,_)|YS],[X1|XS]):-
        todasParagens(YS,XS).
```

Figura 2.8: Predicado que retorna todos os Ids das paragens.

```
agrupaParagem([],[]).
agrupaParagem([paragem(Id,Lat,Long,Estado,TipoAbrigo,
            AbrigoPub,Operadora,Carr,CodRua,NomeRua,Freguesia)],R).
agrupaParagem(
    [paragem(Id,Lat,Long,Estado,TipoAbrigo,AbrigoPub,
            Operadora,Carr,CodRua,NomeRua,Freguesia),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
            Operadora1,Carr,CodRua1,NomeRua1,Freguesia1)|YS],
    [viajar(paragem(Id,Lat,Long,Estado,TipoAbrigo,AbrigoPub,
            Operadora,Carr,CodRua,NomeRua,Freguesia),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
                    Operadora1,Carr,CodRua1,NomeRua1,Freguesia1))|R]) :-
                    agrupaParagem(
                        [paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,
                            AbrigoPub1,Operadora1,Carr,CodRua1,NomeRua1,
                            Freguesia1)|YS],R).
    agrupaParagem(
        [paragem(Id,Lat,Long,Estado,TipoAbrigo,AbrigoPub,
                Operadora,Carr,CodRua,NomeRua,Freguesia),
            paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,
                    Operadora1,Carr1,CodRua1,NomeRua1,Freguesia1)|YS], R) :-
                    agrupaParagem(
                        [paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,
                            AbrigoPub1,Operadora1,Carr1,CodRua1,NomeRua1,
                            Freguesia1)|YS],R).
```

Figura 2.9: Predicado que cria as viagens, ou seja, os vértices do grafo.

2.3 Base de Conhecimento

A base de conhecimento, está apta para representar conhecimento positivo sobre as paragens e o grafo.

```
paragem(815, -104101.68, -97408.6, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(814, -104114.84, -97401.12, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(789, -103478.11, -97851.67, 'Bom', 'Sem Abrigo', 'No', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(169, -103468.05, -97872.21, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(158, -102845.12, -97961.08, 'Bom', 'Sem Abrigo', 'No', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(157, -102859.54, -97965.24, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', [102], 1214,
'Estrada de Queluz', 'Carnaxide e Queijas').
paragem(223, -104280.83, -98312.61, 'Bom', 'Fechado dos Lados', 'No', 'LT', [102, 103],
1766, 'Praceta Antonio Leal de Oliveira', 'Carnaxide e Queijas').
paragem(1009, -104303.63612383851, -98554.7783833525, 'Bom', 'Sem Abrigo', 'No', 'LT',
[102, 108], 79, 'Rua dos Açores', 'Carnaxide e Queijas').
paragem(813, -104117.95, -97049.09, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', [102, 108],
303, 'Rua Cinco de Outubro', 'Carnaxide e Queijas').
paragem(236, -104266.39, -96923.24, 'Bom', 'Sem Abrigo', 'No', 'LT', [102, 108], 308,
'Estrada do Desvio', 'Carnaxide e Queijas').
paragem(817, -104091.69, -96778.69, 'Bom', 'Fechado dos Lados', 'No', 'LT', [102, 108],
362, 'Largo da Patria Nova', 'Carnaxide e Queijas').
```

Figura 2.10: Exemplo de conhecimento perfeito para as Paragens.

2.4 Predicados que calculam a duração da viagem

Nesta sub-secção são descritos todos os predicados que calculam a duração da viagem.

```
funcionamento(6,0,24,0).
```

Figura 2.11: Predicado que indica o horário de funcionamento.

```
hora_ponta(13,0,14,0).
hora_ponta(17,30,18,30).
```

Figura 2.12: Predicado que indica a hora de ponta.

```
hora_atual(6,39).
```

Figura 2.13: Predicado que indica a hora atual.

```
calcTempoViagem([],0).
calcTempoViagem(L,Res):-
        length(L,TAM),Tp is TAM*2,
        verificaAbertura(sim),
        agrupaParagem2(L,Via),
        coef_hora_ponta(Min),
        calDistViag(Via,0,R1),
        Res is round( Tp + (Min + R1)).%,write(R1).
calcTempoViagem(L,fechado).
```

Figura 2.14: Predicado que calcula o tempo entre paragens em minutos.

```
coef_hora_ponta(0):-
    findall(hora_ponta(H,M,H1,M1),
            hora_ponta(H,M,H1,M1),[S,S1]),
            bagof((H3,M3),hora_atual(H3,M3),[(H_a,M_a)]),
            nao(processaL(H_a,M_a,[S,S1],X)).
coef_hora_ponta(MIN):-
    findall(hora_ponta(H,M,H1,M1),
            hora_ponta(H,M,H1,M1),[S,S1]),
            bagof((H3,M3),hora_atual(H3,M3),[(H_a,M_a)]),
            processaL(H_a,M_a,[S,S1],AUX),
            coef(AUX,MIN).
```

Figura 2.15: Predicado que retorna os minutos em espera devido ao transito, ou seja, a hora de ponta.

```
processaL(H,M,[hora_ponta(H1,M1,H2,M2),hora_ponta(H3,M3,H4,M4)],Aux):-
    min1(H,M,X),min1(H1,M1,X1),  Aux is (X1-X) , Aux>= 0,Aux=<60.
processaL(H,M,[hora_ponta(H1,M1,H2,M2),hora_ponta(H3,M3,H4,M4)],Aux):-
    min1(H,M,X),min1(H2,M2,X1),  Aux is (X1-X) , Aux>= 0,Aux=<60.
processaL(H,M,[hora_ponta(H1,M1,H2,M2),hora_ponta(H3,M3,H4,M4)],Aux):-
    min1(H,M,X),min1(H3,M3,X1),  Aux is (X1-X) , Aux>= 0,Aux=<60.
processaL(H,M,[hora_ponta(H1,M1,H2,M2),hora_ponta(H3,M3,H4,M4)],Aux):-
    min1(H,M,X),min1(H4,M4,X1),  Aux is (X1-X) , Aux>= 0,Aux=<60.
-processaL(H,M,[hora_ponta(H1,M1,H2,M2),hora_ponta(H3,M3,H4,M4)],-1).
```

Figura 2.16: Predicado que calcula o valor em minutos entre a hora atual e a hora de ponta.

```
min1(H1,M1,X):- X is (H1*60 + M1).
```

Figura 2.17: Predicado que tranforma as horas em minutos.

```
coef(X,R):- X =< 30, R is (X*0.5).
coef(X,R):- R is ((60-R) * 0.5).
```

Figura 2.18: Predicado correspondente ao coeficiente que determina os minutos em espera, X esta entre 0 e 60 sempre.Tempo máximo em espera é 15 min por paragem.

```
verificaAbertura(sim):-
        bagof((H3,M3),hora_atual(H3,M3),[(H_a,M_a)]),
                bagof((A,B,C,D),funcionamento(A,B,C,D),[(H,M,H1,M1)]),
            min1(H_a,M_a,MM),
                min1(H,M,Fi),min1(H1,M1,FF),
                MM>=Fi  , MM=<FF.
verificaAbertura(nao).
```

Figura 2.19: Predicado que verifica se está na hora de abertura.

```
agrupaParagem2([],[]).
agrupaParagem2([X],[]).
agrupaParagem2([X,Y|XS],[viajar(X,Y)|VS]):-agrupaParagem2(XS,PS).
```

Figura 2.20: Predicado que agrupa paragens em viagens.

```
mudaHora(H,M):- H < 24,H>=0, M<60,M>=0,
        involucao(hora_atual(X,Y)),
        evolucao(hora_atual(H,M)).
mudaHora(H,M):- write('Insira uma hora possivel!'),!,fail.
```

Figura 2.21: Predicado que muda a hora atual.

2.5 Pesquisa no Grafo

Nesta sub-secção procedi a implementação na pesquisas na base de conhecimento,funções de heurística e remoções no grafo.

```
heur1(L,P1,P2,PD,P2):- getId(P1,-1).
heur1(L,P1,P2,PD,P1):- getId(P2,-1).
heur1(L,P1,P2,PD,P1):- getId(P2,Id2),memberchk(Id2,L).
heur1(L,P1,P2,PD,P2):- getId(P1,Id1),memberchk(Id1,L).
heur1(L,P1,P2,PD,P2):- getId(P1,X),getId(P2,X).
heur1(L,P1,P2,PD,P1):- calDist(viajar(P1,PD),R1),
            calDist(viajar(P2,PD),R2) , R1<R2.
heur1(L,P1,P2,PD,P2):- calDist(viajar(P1,PD),R1),
            calDist(viajar(P2,PD),R2) , R1>=R2.
```

Figura 2.22: Predicado que calcula o vértice mais perto (distancia em linha reta) do destino passando como parâmetro as paragens adjacentes nos 2 sentidos.

```
calDist(viajar(
    paragem(-1,Lat,Long,Estado,TipoAbrigo,AbrigoPub,Operadora,Carr,
            CodRua,NomeRua,Freguesia),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,Operadora1,
            Carr1,CodRua1,NomeRua1,Freguesia1)),R):-
                R is sqrt(100234102341).
calDist(viajar(
    paragem(Id,Lat,Long,Estado,TipoAbrigo,AbrigoPub,Operadora,Carr,
            CodRua,NomeRua,Freguesia),
    paragem(-1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,Operadora1,
            Carr1,CodRua1,NomeRua1,Freguesia1)),R):-
                R is sqrt(100234102341).
calDist(viajar(
    paragem(Id,Lat,Long,Estado,TipoAbrigo,AbrigoPub,Operadora,Carr,
            CodRua,NomeRua,Freguesia),
    paragem(Id1,Lat1,Long1,Estado1,TipoAbrigo1,AbrigoPub1,Operadora1,
            Carr1,CodRua1,NomeRua1,Freguesia1)),R) :-
    R is sqrt(((Lat -Lat1) * (Lat -Lat1)) + ((Long-Long1)*(Long-Long1))).
```

Figura 2.23: Predicado que calcula a distancia de uma viagem.

```
encontraCaminho(_, _, paragem(Id1, _, _, _, _, _, _, _, _, _), paragem(Id1, _, _, _,
_, _, _, _, _, _, _), _, R). % cheguei ao destino.
encontraCaminho(_, _, paragem(-1, _, _, _, _, _, _, _, _, _), paragem(Id1, _, _, _, _,
_, _, _, _, _, _), _, []).% qd remove no arestas do grafo,  necessito deste caso de
paragem.
encontraCaminho(L_adj, TodasPar,
        paragem(Id1, Lat1, Long1, Estado1, TipoAbrigo1, AbrigoPub1, Operadora1, Carr1,
        CodRua1, NomeRua1, Freguesia1),
        paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2, Operadora2, Carr2,
        CodRua2, NomeRua2, Freguesia2), P, [IDf|XS]):-
        obterCarr(Id1, TodasPar, C_ori), obterCarr(Id2, TodasPar, C_des),  % Obtem as
        carreiras.
        pri_Comum(C_ori, C_des, NewC),
                % Existe uma carreira em comum.
        NewC \= nao,
        mudarCarreira(NewC, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), Prox2),
        getAdj(L_adj, Id1, NewC, Par_ant, Par_seg),
                                                % Obtem as paragens adjacentes.
        upOrDown(L_adj, Id1, Id2, NewC, A, B),
                                                                % o A, B são posicoes da
        lista de adj.
        integer(A), integer(B),
        ( A<B -> (getId(Par_seg, IDf) , print(Par_seg), print('\n'),  insere(IDf, P, Pf),
        encontraCaminho(L_adj, TodasPar, Par_seg, Prox2, Pf, XS))
        ; (getId(Par_ant, IDf) , print(Par_ant), print('\n'),  insere(IDf, P, Pf),
        encontraCaminho(L_adj, TodasPar, Par_ant, Prox2, Pf, XS)) ).
```

Figura 2.24: Algoritmos 1 de que realiza a pesquisa no grafo.

```
encontraCaminho(L_adj, TodasPar,
    paragem(Id1, Lat1, Long1, Estado1, TipoAbrigo1, AbrigoPub1, Operadora1, Carr1,
    CodRua1, NomeRua1, Freguesia1),
    paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2, Operadora2, Carr2,
    CodRua2, NomeRua2, Freguesia2), P, [IDf|XS]):-
        obterCarr(Id1, TodasPar, C_ori), obterCarr(Id2, TodasPar, C_des),  % Obtem as
        carreiras.
        pri_Comum(C_ori, C_des, nao),% Não existem carreiras em comum
        toList(C_ori, C_ori1), % garante que seja uma lista,  pois pode ser um inteiro.
        map2(P, L_adj, Id1, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), C_ori1, Res),
        best_Choice(P, Res, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), P_Adj_Final),
        getId(P_Adj_Final, IDf), insere(IDf, P, Pf),
        print(P_Adj_Final), print('\n'),
        encontraCaminho(L_adj, TodasPar, P_Adj_Final, paragem(Id2, Lat2, Long2, Estado2,
        TipoAbrigo2, AbrigoPub2, Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), Pf,
        XS).
```

Figura 2.25: Continuação do predicado anterior.

```
encontraCaminho2(_, _, paragem(Id1, _, _, _, _, _, _, _, _, _, _), paragem(Id1, _, _, _,
_, _, _, _, _, _, _), _, R). % cheguei ao destino.
encontraCaminho2(_, _, paragem(-1, _, _, _, _, _, _, _, _, _, _), paragem(Id1, _, _, _,
_, _, _, _, _, _, _), _, []). % Nao existe caminho.
encontraCaminho2(L_adj, TodasPar,
    paragem(Id1, Lat1, Long1, Estado1, TipoAbrigo1, AbrigoPub1, Operadora1, Carr,
    CodRua1, NomeRua1, Freguesia1),
        paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2, Operadora2, Carr,
        CodRua2, NomeRua2, Freguesia2), P, [IDf|XS]):-
        integer(Carr),
        getAdj(L_adj, Id1, Carr, Par_ant, Par_seg),% Obtem as paragens adjacentes.
        upOrDown(L_adj, Id1, Id2, Carr, A, B), % o A, B são posicoes da lista de adj.
        integer(A), integer(B),
        ( A<B -> (getId(Par_seg, IDf) , print(Par_seg), print('\n'),  insere(IDf, P, Pf),
        encontraCaminho2(L_adj, TodasPar, Par_seg, paragem(Id2, Lat2, Long2, Estado2,
        TipoAbrigo2, AbrigoPub2, Operadora2, Carr, CodRua2, NomeRua2, Freguesia2), Pf,
        XS))
        ;(getId(Par_ant, IDf) ,  print(Par_ant), print('\n'),  insere(IDf, P, Pf),
        encontraCaminho2(L_adj, TodasPar, Par_ant, paragem(Id2, Lat2, Long2, Estado2,
        TipoAbrigo2, AbrigoPub2, Operadora2, Carr, CodRua2, NomeRua2, Freguesia2), Pf,
        XS)) ).

encontraCaminho2(L_adj, TodasPar,
    paragem(Id1, Lat1, Long1, Estado1, TipoAbrigo1, AbrigoPub1, Operadora1, Carr1,
    CodRua1, NomeRua1, Freguesia1),
        paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2, Operadora2, Carr2,
        CodRua2, NomeRua2, Freguesia2), P, [IDf|XS]):-
        obterCarr(Id1, TodasPar, C_ori), obterCarr(Id2, TodasPar, C_des), % Obtem as
        carreiras.
        pri_Comum(C_ori, C_des, NewC), % Existe uma carreira em comum.
        NewC \= nao,
        mudarCarreira(NewC, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), Prox2),
        getAdj(L_adj, Id1, NewC, Par_ant, Par_seg), % Obtem as paragens adjacentes.
        upOrDown(L_adj, Id1, Id2, NewC, A, B),
        integer(A), integer(B),
        ( A<B -> (getId(Par_seg, IDf), \+ memberchk(IDf, P) , print(Par_seg),
        print('\n'),  insere(IDf, P, Pf), encontraCaminho2(L_adj, TodasPar, Par_seg,
        Prox2, Pf, XS))
        ; (getId(Par_ant, IDf), \+ memberchk(IDf, P),  print(Par_ant), print('\n'),
        insere(IDf, P, Pf), encontraCaminho2(L_adj, TodasPar, Par_ant, Prox2, Pf, XS)) ).
```

Figura 2.26: Predicado do 2 algoritmo de pesquisa de um caminho.

```
encontraCaminho2(L_adj, TodasPar,
    paragem(Id1, Lat1, Long1, Estado1, TipoAbrigo1, AbrigoPub1, Operadora1, Carr1,
    CodRua1, NomeRua1, Freguesia1),
    paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2, Operadora2, Carr2,
    CodRua2, NomeRua2, Freguesia2), P, [IDf|XS]):-
        obterCarr(Id1, TodasPar, C_ori), obterCarr(Id2, TodasPar, C_des), % Obtem as
        carreiras.
        pri_Comum(C_ori, C_des, nao), % Não existem carreiras em comum
        toList(C_ori, C_ori1),% garante que seja uma lista,  pois pode ser um inteiro.
        map2(P, L_adj, Id1, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), C_ori1, Res),
        best_Choice(P, Res, paragem(Id2, Lat2, Long2, Estado2, TipoAbrigo2, AbrigoPub2,
        Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), P_Adj_Final),
        getId(P_Adj_Final, IDf), \+ memberchk(IDf, P), insere(IDf, P, Pf),
        print(P_Adj_Final), print('\n'),
        encontraCaminho2(L_adj, TodasPar, P_Adj_Final, paragem(Id2, Lat2, Long2, Estado2,
        TipoAbrigo2, AbrigoPub2, Operadora2, Carr2, CodRua2, NomeRua2, Freguesia2), Pf,
        XS).
```

Figura 2.27: Continhução do predicado encontraCaminho2

```
pontos_inter(L,Todas,O,D,[],SOL):- pontos_inter_aux(O,D).
pontos_inter(L,Todas,O,D,[X|XS],SOL):-
 pontos_inter_aux(O,X),write('->Paragem\n'),
 pontos_inter(L,Todas,X,D,XS,SOL).
```

Figura 2.28: Predicado que calcula as viagens para todos os pontos intermédios e finais.

```
pontos_inter_aux(O,D):-bagof(_,grafo(L1,L),[L|R]),
            findall(paragem(Id1,Lat,Long,Estado,TipoAbrigo,AbrigoPub,Operadora,
                    Carr,CodRua,NomeRua,Freguesia),
            paragem(Id1,Lat,Long,Estado,TipoAbrigo,AbrigoPub,Operadora,Carr,CodRua,
                    NomeRua,Freguesia),Todas),
            getParagem(O,Par1),getParagem(D,Par2),
            encontraCaminhoS(L,Todas,Par1,Par2,[O],Caminho),
            printl(Caminho).
```

Figura 2.29: Predicado auxiliar par o calculo das viagens para os pontos intermédios.

```
seleciona_OP_Aux([], L_OP, R).
seleciona_OP_Aux([viajar(X1, X2)|XS], L_OP, [viajar(X1, X2)|PP]) :- getOperadora(X1, A),
getOperadora(X2, B),
    memberchk(A, L_OP), memberchk(B, L_OP),          % são membros
        seleciona_OP_Aux(XS, L_OP, PP).
seleciona_OP_Aux([viajar(X1, X2)|XS], L_OP, PP) :- seleciona_OP_Aux(XS, L_OP, PP).
```

Figura 2.30: Predicado auxiliar para selecionar viagens de operadoras passadas como parâmetro.

```
excluir_OP([], L_OP, R).
excluir_OP([viajar(X1, X2)|XS], L_OP, PP) :- getOperadora(X1, A), getOperadora(X2, B),
    (memberchk(A, L_OP);memberchk(B, L_OP)),
     excluir_OP(XS, L_OP, PP).
excluir_OP([viajar(X1, X2)|XS], L_OP, [viajar(X1, X2)|PP]) :- excluir_OP(XS, L_OP, PP).
```

Figura 2.31: Predicado auxiliar para remover viagens de operadoras passadas como parâmetro.

```
excluir_ABR_SemPub([], []).
excluir_ABR_SemPub([viajar(X1, X2)|XS], Res):- (getAbrPub(X1, 'No');getAbrPub(X2, 'No')),
excluir_ABR_SemPub(XS, Res) .
excluir_ABR_SemPub([viajar(X1, X2)|XS], [viajar(X1, X2)|Res]):- excluir_ABR_SemPub(XS,
Res) .
```

Figura 2.32: Predicado que exclui todas as viagens sem publicidade.

```
seleciona_Abrigadas([], []).
seleciona_Abrigadas([viajar(X1, X2)|XS], Res):- (getTipoAbr(X1, 'Sem Abrigo');
getTipoAbr(X2, 'Sem Abrigo')), seleciona_Abrigadas(XS, Res).
seleciona_Abrigadas([viajar(X1, X2)|XS], [viajar(X1, X2)|Res]):- seleciona_Abrigadas(XS,
Res).
```

Figura 2.33: Predicado que exclui todas as viagens que não são abrigadas.

Capítulo 3

Análise de Resultados

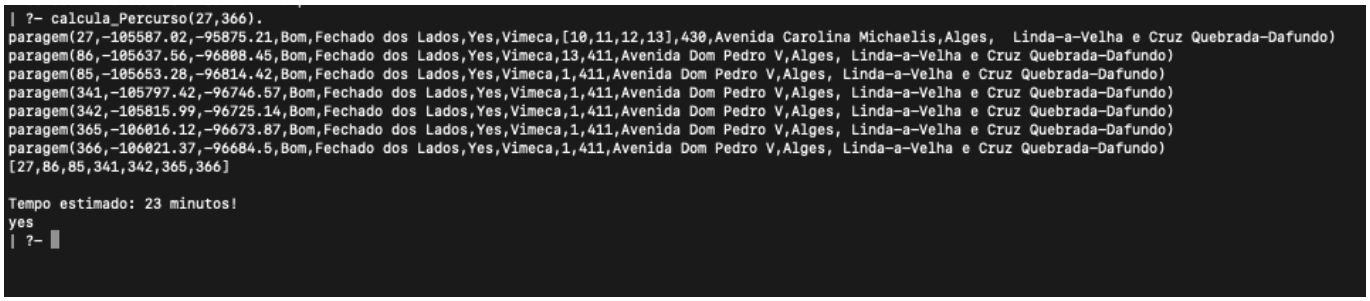

Figura 3.1: Resultado do Predicado *calcula_percurso* que calcula um trajeto entre dois pontos

Figura 3.2: Resultado do Predicado *percurso_seleciona_op* que seleciona apenas algumas das operadoras de transporte para um determinado percurso

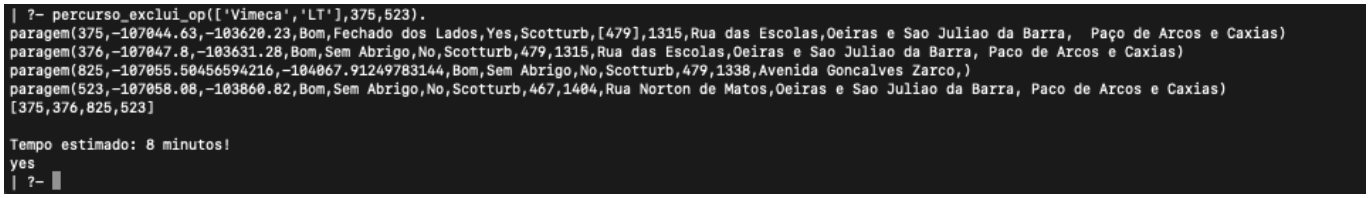

Figura 3.3: Resultado do Predicado $percurso_exclui\_op que excluir um ou mais operadores de transporte para o percurso$

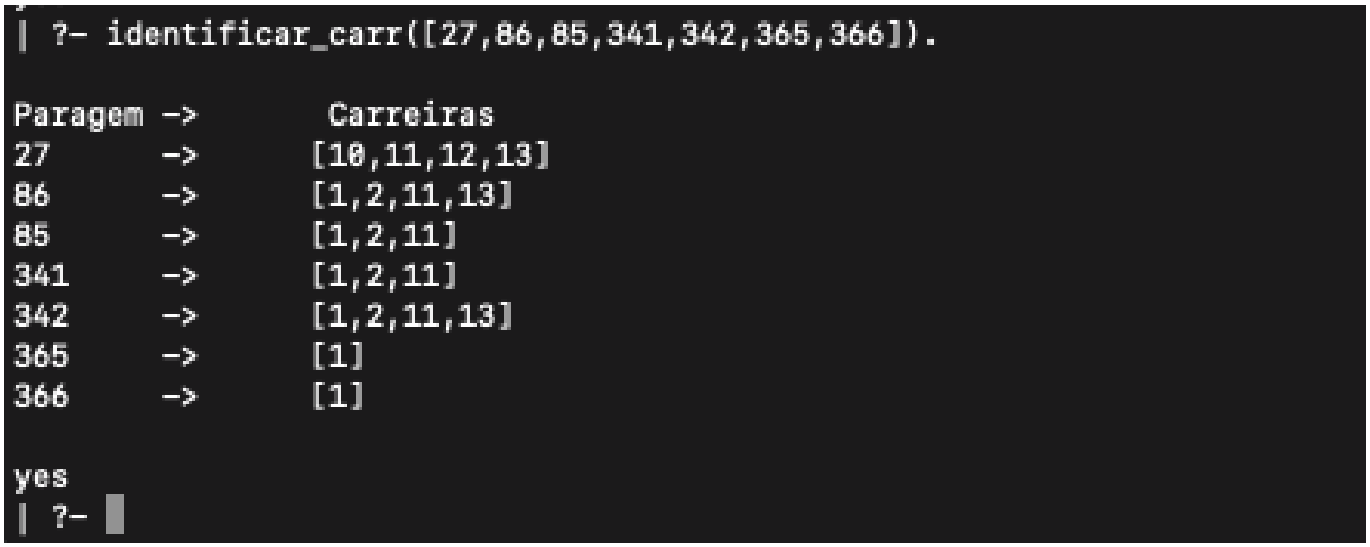

Figura 3.4: Resultado do Predicado *identificar_carr* que identifica as paragens com o maior número de carreiras num determinado percurso.

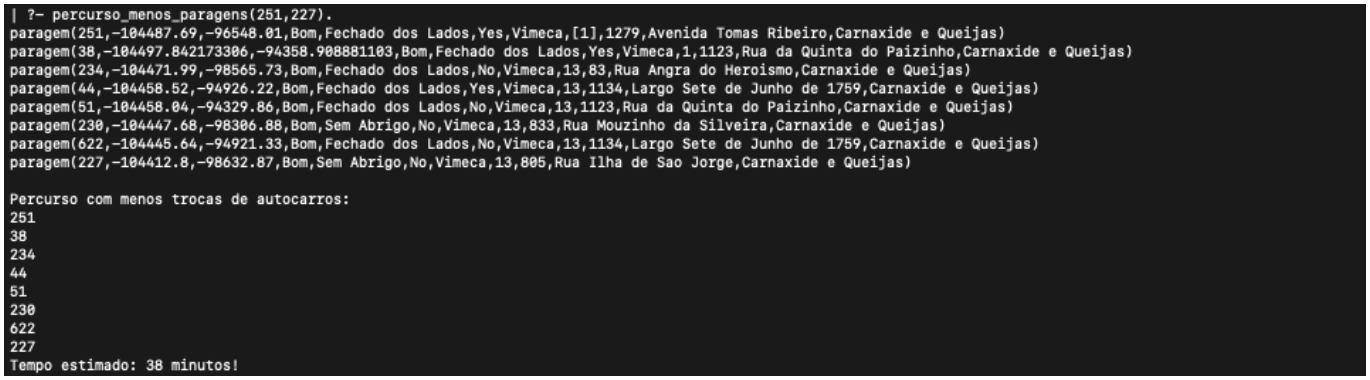

Figura 3.5: Resultado do Predicado *percurso_menos_paragens* que escolhe o menor percurso (usando critério menor número de paragens)

```
| ?- percurso_mais_rapido(251,227).
paragem(251,-104487.69,-96548.01,Bom,Fechado dos Lados,Yes,Vimeca,[1],1279,Avenida Tomas Ribeiro,Carnaxide e Queijas)
paragem(38,-104497.842173306,-94358.908881103,Bom,Fechado dos Lados,Yes,Vimeca,1,1123,Rua da Quinta do Paizinho,Carnaxide e Queijas)
paragem(234,-104471.99,-98565.73,Bom,Fechado dos Lados,No,Vimeca,13,83,Rua Angra do Heroismo,Carnaxide e Queijas)
paragem(230,-104447.68,-98306.88,Bom,Sem Abrigo,No,Vimeca,2,833,Rua Mouzinho da Silveira,Carnaxide e Queijas)
paragem(227,-104412.8,-98632.87,Bom,Sem Abrigo,No,Vimeca,2,805,Rua Ilha de Sao Jorge,Carnaxide e Queijas)

Percurso mais rapido!
251
38
234
230
227
Tempo estimado: 32 minutos!
yes
| ?-
```

Figura 3.6: Resultado do Predicado *percurso_mais_rapido* que escolhe o percurso mais rápido (usando um critério da distância)

```
| ?- percurso_pub(13,667).
paragem(13,-105268.05,-95547.68,Bom,Fechado dos Lados,Yes,Carris,[201,748,751],124,Avenida Jose Gomes Ferreira,Alges,  Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(651,-105300.44,-95336.46,Bom,Fechado dos Lados,Yes,Carris,201,113,Alameda Fernao Lopes,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(15,-105325.87,-95135.44,Bom,Fechado dos Lados,Yes,Carris,201,113,Alameda Fernao Lopes,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(645,-105353.27,-95172.19,Bom,Fechado dos Lados,Yes,Carris,201,113,Alameda Fernao Lopes,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(54,-105456.01,-94993.65,Bom,Fechado dos Lados,Yes,Carris,201,116,Avenida General Norton de Matos,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(53,-105462.27,-94976.17,Bom,Fechado dos Lados,Yes,Carris,201,116,Avenida General Norton de Matos,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(640,-105655.76,-95028.52,Bom,Fechado dos Lados,Yes,Carris,201,116,Avenida General Norton de Matos,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(633,-105696.83,-95075.27,Bom,Fechado dos Lados,Yes,Carris,201,116,Avenida General Norton de Matos,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(677,-106071.42513405527,-95039.14634930693,Bom,Fechado dos Lados,Yes,Carris,201,10,Avenida dos Bombeiros Voluntarios de Alges,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(670,-106112.38652897863,-95027.71017120728,Bom,Fechado dos Lados,Yes,Carris,201,10,Avenida dos Bombeiros Voluntarios de Alges,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(676,-106283.09180093784,-95136.51301607292,Bom,Sem Abrigo,Yes,Carris,201,10,Avenida dos Bombeiros Voluntarios de Alges,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
paragem(667,-106342.5,-95131.58,Bom,Fechado dos Lados,Yes,Carris,201,10,Avenida dos Bombeiros Voluntarios de Alges,Alges, Linda-a-Velha e Cruz Quebrada-Dafundo)
[13,651,15,645,54,53,640,633,677,670,676,667]

Tempo estimado: 26 minutos!
yes
```

Figura 3.7: Resultado do Predicado *percurso_pub* que escolhe o percurso que passe apenas por abrigos com publicidade

```
| ?- percurso_abrigado(795,827).
paragem(795,-104741.4,-101691.52,Bom,Fechado dos Lados,Yes,LT,[106,112,125,129,184],692,Rua Conde de Rio Maior,Porto Salvo)
paragem(796,-104911.86,-101688.38,Bom,Fechado dos Lados,Yes,LT,112,692,Rua Conde de Rio Maior,Porto Salvo)
paragem(828,-105046.86,-101627.86,Bom,Fechado dos Lados,Yes,LT,112,1540,Estrada de Paco de Arcos,Porto Salvo)
paragem(830,-105227.47,-102176.58,Bom,Fechado dos Lados,Yes,LT,112,1682,Avenida Santa Casa da Misericordia de Oeiras,Porto Salvo)
paragem(832,-105236.25,-102190.54,Bom,Fechado dos Lados,Yes,LT,112,1682,Avenida Santa Casa da Misericordia de Oeiras,Porto Salvo)
paragem(827,-105268.41,-102428.49,Bom,Fechado dos Lados,Yes,LT,112,1680,Rua Encosta das Lagoas,Porto Salvo)
[795,796,828,830,832,827]

Tempo estimado: 14 minutos!
yes
| ?-
```

Figura 3.8: Resultado do Predicado *percurso_abrigado* que escolher o percurso que passe apenas por paragens abrigadas.

```
| ?- percurso_inter(251,227,[44,45]).
251
44->Paragem
251
38
620
45->Paragem
620
224
234
230
227
yes
| ?-
```

Figura 3.9: Resultado do Predicado do primeiro percurso intermédio. *percurso_inter*

Figura 3.10: Resultado do Predicado do segundo percurso intermédio. *percurso_inter*

Capítulo 4

Conclusão e trabalho futuro

Nos dias de hoje, devido à população querer que as cidades sejam mais inteligentes, temos um desafio enorme enquanto alunos de engenharia informática. Para isso foi desenvolvido este projeto com objetivo de melhorar a experiência de utilização dos transportes públicos.
Começando por dizer que neste trabalho apenas é representado conhecimento perfeito, que pode ser positivo e negativo. Este é usado na representação das paragens e do grafo, que nos permite saber quais os caminhos existentes entre paragens, ambos obtidos através de ficheiros csv. De seguida, implementei alguns sistemas de pesquisa no grafo e o calculo do tempo de um certa paragem. Uma melhoria neste sistema seria a colocação de acidentes , visto que, infelizmente existem vários por dia.
Conclui-se que, o trabalho foi bem desenvolvido, tendo contribuído para melhorar o conhecimento obtido nas aulas e até mesmo obter novo conhecimentos na representação de grafos com programação lógica.

Capítulo 5

Anexos

Esta função foi usada para garantir que todas as paragens do grafo estavam incluídas na base de conhecimento.

```
int []  a1 = {...} ; //preenchidos com todos os nós do ficheiro exel das
↪  paragens
int []  a2 = {...} ; // todos os nós do grafo, ou seja a lista de adj.
int []aux = new int[100];// array onde vão ficar as paragens que não existem no
↪  grafo
int i=0; // numero de elementos do array aux.
for(int e1:a1){
    boolean p= false;
    for(int e2:a2) {
        if (e1 == e2) {
            p=true;
            break;
        }
    }
    if (p==false)
        aux[i++]=e1;
}
```

Figura 5.1: Função usada para eliminar incongruências nos dados.

A diferença dos dois ficheiros **csv** está na lista de carreiras, ou seja, a rota que os autocarros percorrem.

```
  183; -103678.36; -96590.26; Bom; Fechado dos Lados; Yes; Vimeca; 01; 286; Rua Aquilino
  Ribeiro; Carnaxide e Queijas
791; -103705.46; -96673.6; Bom; Aberto dos Lados; Yes; Vimeca; 01; 286; Rua Aquilino
Ribeiro; Carnaxide e Queijas
595; -103725.69; -95975.2; Bom; Fechado dos Lados; Yes; Vimeca; 01; 354; Rua Manuel
Teixeira Gomes; Carnaxide e Queijas
```

Figura 5.2: Exemplo do ficheiro csv da lista de adjacência.

```
185; -103922.82; -96235.62; Bom; Fechado dos Lados; Yes; Scotturb; 01,02,07,10,12,13,15;
354; Rua Manuel Teixeira Gomes; Carnaxide e Queijas
250; -104031.08; -96173.83; Bom; Fechado dos Lados; Yes; Vimeca; 01,02,07,10,12,13,15;
1113; Avenida de Portugal; Carnaxide e Queijas
107; -103972.32; -95981.88; Bom; Fechado dos Lados; Yes; Vimeca; 01,02,07,10,12,13,15;
1113; Avenida de Portugal; Carnaxide e Queijas
```

Figura 5.3: Exemplo do ficheiro csv que serve para a descrição da paragem.