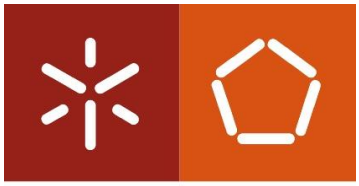


Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre - 2019/20)

Relatório SRCR - TP Individual

A85919 Pedro Dias Parente

Braga

5 de Junho de 2020

Resumo

Este relatório tem como objetivo descrever a metodologia usadada para resolver a problemática proposta pelos docentes da cadeira de Sistemas de Representação de Conhecimento e Raciocínio.

O processo consistiu na construção de um conjunto de regras e predicados que, a partir de uma grande quantidade de informação sobre viagens de autocarro no concelho de Oeiras, fosse capaz de indicar uma rota possível que respeita-se as restrições impostas. Com este intuito, usei então o meu conhecimento de Prolog para tentar criar um conjunto de "queries"que fossem capaz de responder a cada problema apresentado no enunciado.

Conteúdo

1 Introdução **5**

2 Descrição do Trabalho e Análise de Resultados **6**
2.1 Base de Conhecimento 7
2.2 Sistema de inferência 9

3 Conclusão **18**

4 Referências **19**
4.1 Referências Bibliográficas 19
4.2 Referências Eletrónicas 20

Lista de Figuras

2.1 Código feito para a construção do Parser 8
2.2 Predicado calcula_trajeto 9
2.3 Exemplo demorado de calcula_trajeto - demora cerca de 10 segundos 9
2.4 Predicado calcula_melhor_trajeto 10
2.5 Exemplo de calcula_melhor_trajeto - a calcula_trajeto não consegue descobrir uma rota entre esta origem e destino em tempo razoável 10
2.6 Predicado calcula_trajeto_operadora 11
2.7 Predicado calcula_trajeto_sem_operadora 11
2.8 Predicado calcula_maior_carreiras 12
2.9 Exemplo calcula_maior_carreiras 12
2.10 Predicado calcula_trajeto_publicidade 13
2.11 Predicado calcula_trajeto_abrigados 13
2.12 Predicado calcula_trajeto_intermedios 14
2.13 Exemplo calcula_trajeto_intermedios - em vez de utilizar a rota mais "normal"para chegar ao destino, faz um desvio e passa pelos pontos intermédios 14
2.14 Predicado resolve_informado_distancia 15
2.15 Resultado resolve_informado_distancia - retorna em conjunto com o caminho a distância percorrida 15
2.16 Predicado resolve_informado_paragens 16
2.17 Resultado resolve_informado_paragens - Comparativamente ao calcula_trajeto, devolveu um caminho muito mais pequeno 16
2.18 Predicados Auxiliares 17

Capítulo 1

Introdução

A finalidade deste trabalho, como referido anteriormente, foi a representação do universo de viagens de autocarro no conselho de Oeiras, recorrendo, para esse fim, à programação em Prolog.

A equipa docente forneceu-nos com uma "base de dados"em excel, que descreve as paragens existentes e as ligações entre paragens adjacentes.

Este universo consistirá, assim, na existência de paragens (nodos) e ligações entre essas paragens, como arestas em grafos. É de realçar que apenas trabalhei com a informação disponibilizada pelos docentes, não extendendo o conhecimento, achando que a quantidade de informação disponível ja era suficiente.

Capítulo 2

Descrição do Trabalho e Análise de Resultados

O trabalho está divido em duas partes.

A primeira componente será a **base de conhecimento**, onde será guardado o conhecimento sobre todas as paragens e as consequentes ligações entre elas.

A outra componente, o **sistema de inferência** irá conter os predicados responsáveis por responder aos problemas propostos no trabalho, como por exemplo, encontrar uma rota entre uma origem e um destino onde apenas se passe por paragens com publicidade.

2.1 Base de Conhecimento

Como referido, a base de conhecimento foi construída a partir dos ficherios excel fornecidos pela equipa docente, porém, estes não se encontravam de uma forma possível de utilizar diretamente pelo prolog.

Então, com o intuito de construir uma base de informação consistente e que fosse capaz de ser analisada no programa, foi primeiro feito um ***parser*** que lesse os ficheiros e escrevesse num outro ficheiro (**base_de_conhecimento.pl**) factos como:

- paragem(45,-104578.88,-94652.12,'Bom','Sem Abrigo','No','Vimeca',10,365,'Estrada da Portela','Carnaxide e Queijas').

  para indicar que existe uma paragem com estas caraterísticas.

- adjacente(paragem(183,-103678.36,-96590.26,'Bom','Fechado dos Lados','Yes','Vimeca',1,286,'Rua Aquilino Ribeiro','Carnaxide e Queijas'),paragem(791,-103705.46,-96673.6,'Bom','Aberto dos Lados','Yes','Vimeca',1,286,'Rua Aquilino Ribeiro','Carnaxide e Queijas')).

  para indicar que as duas paragens estão adjacentes e que há, por consequência, uma ligação entre elas.

A paragem é careterizada por um ID, latitude, longitude, Estado de Conservação, Tipo de Abrigo, ter ou não publicidade, a Operadora, Carreira, Código de Rua e, Nome da Rua e Freguesia.
A parser foi escrito em java e baseava-se em duas regras:

- No ficheiro lista_adjacencias_paragens.csv paragens seguidas era adjacentes;
- A exceção à regra anterior é de quando muda de carreira.

Sabendo isto com um simples programa fui capaz de gerar o ficheiro base_de_conhecimento.pl e passar à próxima etapa.

Figura 2.1: Código feito para a construção do Parser

```
public class Main {
    public static void main(String[] args) {
        try {
            FileReader reader = new FileReader( fileName: "prolog/lista_adjacencias_paragens.csv");

            try (BufferedReader br = new BufferedReader(reader)) {
                ArrayList<String> lines = new ArrayList<>();
                String parts1[];
                String parts2[];
                String line;
                FileWriter writerFile = new FileWriter( fileName: "prolog/base_de_conhecimento.pl");

                while ((line = br.readLine()) != null) {
                    lines.add(line);
                }
                reader.close();
                writerFile.write( str: ":- dynamic adjacente/2.\n");
                writerFile.write( str: ":- dynamic paragem/11.\n");
                for(int i = 1; i < lines.size() - 1; i++){
                    parts1 = lines.get(i).split( regex: ";");
                    parts2 = lines.get(i+1).split( regex: ";");
                    if (parts1[7].equals(parts2[7])){
                        writerFile.write( str: "adjacente(");
                        writerFile.write( str: "paragem(" + parts1[0] + "," + parts1[1] + "," + parts1[2] + ",'" + parts1[
                        writerFile.write( str: "paragem(" + parts2[0] + "," + parts2[1] + "," + parts2[2] + ",'" + parts2[
                        writerFile.write( str: ").\n");
                        writerFile.write( str: "adjacente(");
                        writerFile.write( str: "paragem(" + parts2[0] + "," + parts2[1] + "," + parts2[2] + ",'" + parts2[
                        writerFile.write( str: "paragem(" + parts1[0] + "," + parts1[1] + "," + parts1[2] + ",'" + parts1[
                        writerFile.write( str: ").\n");
                    }
                }

                for(int i = 1; i < lines.size() - 1; i++){
                    parts1 = lines.get(i).split( regex: ";");
                    writerFile.write( str: "paragem(" + parts1[0] + "," + parts1[1] + "," + parts1[2] + ",'" + parts1[3] +
                }
                parts2 = lines.get(lines.size()-1).split( regex: ";");
                writerFile.write( str: "paragem(" + parts2[0] + "," + parts2[1] + "," + parts2[2] + ",'" + parts2[3] + "',
                writerFile.close();
            }
        } catch (IOException e) {
            System.out.println("Erro a ler/escrever ficheiro");
```

2.2 Sistema de inferência

Como foi referido anteriormente, foi desenvolvido um conjunto de predicados capazes de responder aos problemas propostos:

O **primeiro** destes predicados calcula um trajeto possível entre dois pontos, uma origem e um destino, e foi usado uma adaptação do algoritmo depth first.

Figura 2.2: Predicado calcula_trajeto

```
calcula_trajeto(Origem,Dest,Caminho):-
            profundidade(Origem,Dest,[Origem],Caminho).

profundidade(Dest, Dest, H, C):- inverso(H,C).

profundidade(Origem, Dest, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr1,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr2,_,_,_)),
            \+ member(Prox, His),
            profundidade(Prox,Dest,[Prox|His],C).
```

Este predicado funciona para maiorparte dos casos instantaneamente mas é importante realçar que em alguns casos demora algum tempo devido à própria natureza do algoritmo depth first e à quantidade de informação disponível.

Figura 2.3: Exemplo demorado de calcula_trajeto - demora cerca de 10 segundos

```
| ?- calcula_trajeto(97,526,R).
R = [97,98,99,707,717,716,726,727,722,723,751,780,732,152,136,135,744,134,715,1
27,125,711,104,118,120,121,122,141,143,142,139,140,137,138,770,767,779,775,776,
774,773,777,778,762,756,781,785,208,797,191,795,796,828,284,285,282,283,281,294
,378,379,989,869,88,419,541,542,503,516,543,10,540,554,552,553,565,551,564,570,
549,548,399,398,875,874,390,408,404,403,394,873,407,871,391,872,421,442,411,437
,431,274,273,326,327,318,319,317,325,324,320,838,836,835,837,842,841,810,811,80
2,804,632,803,229,800,809,808,805,801,1009,223,236,816,235,813,814,815,817,692,
693,178,947,792,710,807,818,823,952,954,1002,977,986,983,354,353,863,856,857,36
7,333,846,845,330,364,33,32,60,61,64,63,62,58,57,59,654,655,56,491,490,458,457,
335,363,344,345,343,346,342,341,85,86,347,339,334,357,336,610,611,613,612,49,48
,601,602,600,42,46,614,45,620,38,251,44,51,622,254,249,608,985,40,599,50,39,628
,604,255,242,609,953,597,261,250,107,89,185,594,180,181,593,499,182,595,791,183
,170,788,68,1014,709,154,87,687,686,742,741,155,159,734,156,161,162,172,171,799
,1010,246,260,227,230,234,224,239,238,226,1001,607,232,52,233,231,241,240,859,8
58,332,861,359,349,29,646,642,30,17,643,20,36,638,637,361,362,37,26,27,28,641,6
35,679,688,675,72,75,671,657,70,526] ?
```

O **segundo** predicado também calcula um trajeto possível entre dois pontos, uma origem e um destino, mas desta vez foi tentado utilizar a informação relativa às carreiras para otimizar o algoritmo anterior. Este funciona de seguinte maneira:

- Primeiro testa se já está na carreira destino, e, se sim, usa o algoritmo depth first para chegar ao destino rapidamente.
- Se não, vê se algum dos seus próximos nodos vai ter como carreira a carreira destino e se sim, usa esse nodo como próximo nodo até chegar ao destino.

Figura 2.4: Predicado calcula_melhor_trajeto

```
calcula_melhor_trajeto(Origem,Dest,CarrD,Caminho):-
            paragem(Origem,_,_,_,_,_,_,CarrO,_,_,_),
            paragem(Dest,_,_,_,_,_,_,CarrD,_,_,_),
            CarrO = CarrD,
            profundidade_melhor(Origem,Dest,[Origem],Caminho).

profundidade_melhor(Dest, Dest, H, C):- inverso(H,C).

profundidade_melhor(Origem, Dest, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr1,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr2,_,_,_)),
            \+ member(Prox, His),
            profundidade_melhor(Prox,Dest,[Prox|His],C).

calcula_melhor_trajeto(Origem,Dest,CarrD,Caminho):-
            paragem(Origem,_,_,_,_,_,_,CarrO,_,_,_),
            paragem(Dest,_,_,_,_,_,_,CarrD,_,_,_),
            CarrO \= CarrD,
            profundidade_melhor_carr(Origem,Dest,[Origem],Caminho).

profundidade_melhor_carr(Dest, Dest, H, C):- inverso(H,C).

profundidade_melhor_carr(Origem, Dest, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr1,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr2,_,_,_)),
            paragem(Dest,_,_,_,_,_,_,CarrD,_,_,_),
            paragem(Prox,_,_,_,_,_,_,CarrD,_,_,_),
            \+ member(Prox,His),
            profundidade_melhor_carr(Prox,Dest,[Prox|His],C).

profundidade_melhor_carr(Origem, Dest, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr1,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr2,_,_,_)),
            paragem(Dest,_,_,_,_,_,_,CarrD,_,_,_),
            paragem(Prox,_,_,_,_,_,_,CarrP,_,_,_),
            CarrP \= CarrD,
            profundidade_melhor_carr(Prox,Dest,[Prox|His],C).
```

Este predicado consegue chegar a alguns destinos que o predicado anterior não conseguiria pois é informado, porém continua a não ser 100% eficaz porque ainda tem uma natureza de depth first por baixo.

Figura 2.5: Exemplo de calcula_melhor_trajeto - a calcula_trajeto não consegue descobrir uma rota entre esta origem e destino em tempo razoável

```
| ?-
| ?- calcula_melhor_trajeto(183,745,2,R).
R = [183,791,595,182,181,180,594,185,89,107,250,261,597,953,609,242,255,604,628
,39,50,599,40,985,608,249,254,622,51,44,234,230,227,260,246,1010,799,171,172,16
2,161,734,156,147,736,745] ?
```

O **terceiro** predicado responde à query de retornar uma rota onde apenas se ande de autocarro com uma Operadora que pertença às Operadoras dadas como argumento. Também é utilizado o algoritmo depth first.

Figura 2.6: Predicado calcula_trajeto_operadora

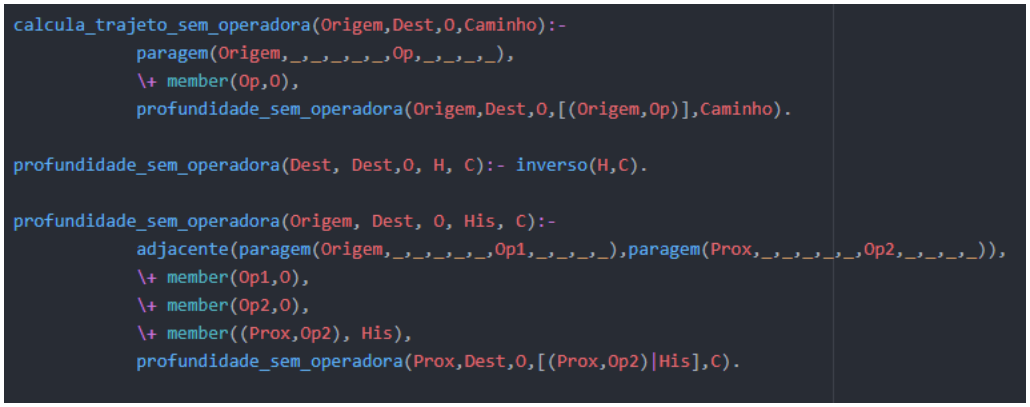
```
calcula_trajeto_operadora(Origem,Dest,O,Caminho):-
            paragem(Origem,_,_,_,_,_,Op1,_,_,_,_),
            member(Op1,O),
            paragem(Dest,_,_,_,_,_,Op2,_,_,_,_),
            member(Op2,O),
            profundidade_operadora(Origem,Dest,O,[(Origem,Op1)],Caminho).

profundidade_operadora(Dest, Dest,O, H, C):- inverso(H,C).

profundidade_operadora(Origem, Dest, O, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,Op1,_,_,_,_),paragem(Prox,_,_,_,_,_,Op2,_,_,_,_)),
            member(Op1,O),
            member(Op2,O),
            \+ member((Prox,Op2), His),
            profundidade_operadora(Prox,Dest,O,[(Prox,Op2)|His],C).
```

O **quarto** predicado é extremamente similar ao anterior, só que em vez de fornecermos um conjunto de Operadores as quais queremos que a Operadora do autocarro pertença, damos um conjunto de Operadoras as quais queremos que **não pertença**.

Figura 2.7: Predicado calcula_trajeto_sem_operadora



```
calcula_trajeto_sem_operadora(Origem,Dest,O,Caminho):-
            paragem(Origem,_,_,_,_,_,Op,_,_,_,_),
            \+ member(Op,O),
            profundidade_sem_operadora(Origem,Dest,O,[(Origem,Op)],Caminho).

profundidade_sem_operadora(Dest, Dest,O, H, C):- inverso(H,C).

profundidade_sem_operadora(Origem, Dest, O, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,Op1,_,_,_,_),paragem(Prox,_,_,_,_,_,Op2,_,_,_,_)),
            \+ member(Op1,O),
            \+ member(Op2,O),
            \+ member((Prox,Op2), His),
            profundidade_sem_operadora(Prox,Dest,O,[(Prox,Op2)|His],C).
```

O **quinto** predicado é vai retornar uma lista ordenada das paragens com o maior número de carreiras num determinado percurso. Este foi atingido com um predicado auxiliar que retorna o número de carreiras por ID, e um algoritmo quick sort que me ordena posteriormente a lista com os túpulos (ID,nº de Carreiras).

Figura 2.8: Predicado calcula_maior_carreiras

```
calcula_maior_carreiras(Origem,Dest,L):-
            numero_paragens(Origem,Len),
            profundidade_maior_carreiras(Origem,Dest,[(Origem,Len)],Lista),
            quick_sort(Lista,L).

profundidade_maior_carreiras(Dest, Dest, H, L):- inverso(H,L).

profundidade_maior_carreiras(Origem, Dest, His, L):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr1,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr2,_,_,_)),
            numero_paragens(Prox,Len),
            \+ member((Prox,Len),His),
            profundidade_maior_carreiras(Prox,Dest,[(Prox,Len)|His],L).


numero_paragens(Num,Len):-
            findall(Num, paragem(Num,_,_,_,_,_,_,_,_,_,_), Lista),
            length(Lista,Len).

%-----------------------------------------------

quick_sort(List,Sorted):-q_sort(List,[],Sorted).
q_sort([],Acc,Acc).
q_sort([H|T],Acc,Sorted):-
  pivoting(H,T,L1,L2),
  q_sort(L1,Acc,Sorted1),q_sort(L2,[H|Sorted1],Sorted).

pivoting((N,H),[],[],[]).
pivoting((N,H),[(Y,X)|T],[(Y,X)|L],G):-X=<H,pivoting((N,H),T,L,G).
pivoting((N,H),[(Y,X)|T],L,[(Y,X)|G]):-X>H,pivoting((N,H),T,L,G).
```

Aqui está um exemplo do predicado a retornar exatamente o pedido:

Figura 2.9: Exemplo calcula_maior_carreiras

```
| ?-
| ?- calcula_maior_carreiras(183,597,R).
R = [(595,7),(183,6),(791,6),(594,6),(185,6),(107,6),(250,6),(597,6),(182,5),(1
81,5),(180,5),(89,5),(261,3),(499,1),(593,1)] ?
```

O **sexto** predicado precupa-se com retornar uma rota que apenas passe por paragens que exibem publicidade.

Figura 2.10: Predicado calcula_trajeto_publicidade

```
calcula_trajeto_publicidade(Origem,Dest,Caminho):-
            paragem(Origem,_,_,_,_,Pub1,_,_,_,_,_),
            Pub1 = 'Yes',
            paragem(Dest,_,_,_,_,Pub2,_,_,_,_,_),
            Pub2 = 'Yes',
            profundidade_publicidade(Origem,Dest,[Origem],Caminho).

profundidade_publicidade(Dest, Dest, H, C):- inverso(H,C).

profundidade_publicidade(Origem, Dest, His, C):-
            adjacente(paragem(Origem,_,_,_,_,P1,_,_,_,_,_),paragem(Prox,_,_,_,_,P2,_,_,_,_,_)),
            P2 = 'Yes',
            \+ member(Prox, His),
            profundidade_publicidade(Prox,Dest,[Prox|His],C).
```

O **sétimo** predicado, semelhantemente ao anterior, precupa-se com retornar uma rota que apenas passe por paragens que estejam abrigadas.

Figura 2.11: Predicado calcula_trajeto_abrigados

```
calcula_trajeto_abrigados(Origem,Dest,Caminho):-
           paragem(Origem,_,_,_,Abrig1,_,_,_,_,_,_),
           Abrig1 \= 'Sem Abrigo',
           paragem(Dest,_,_,_,Abrig2,_,_,_,_,_,_),
           Abrig2 \= 'Sem Abrigo',
           profundidade_abrigados(Origem,Dest,[Origem],Caminho).

profundidade_abrigados(Dest, Dest, H, C):- inverso(H,C).

profundidade_abrigados(Origem, Dest, His, C):-
           adjacente(paragem(Origem,_,_,_,Abrig1,_,_,_,_,_,_),paragem(Prox,_,_,_,Abrig2,_,_,_,_,_,_)),
           Abrig2 \= 'Sem Abrigo',
           \+ member(Prox, His),
           profundidade_abrigados(Prox,Dest,[Prox|His],C).
```

O **oitavo** predicado é um pouco mais complexo que os anteriores, pois precupa-se com retornar uma rota que certifique-se que passou por uns pontos intermédios fornecidos como argumento. Para resolver este problema a tática utilizada foi apenas verificar ao fim de encontrar um caminho se a lista de pontos intermedios fornecidos como argumento estavam presentes no Caminho resultante. Porém, como é usado o algoritmo depth first, dependendo da origem e destino fornecido por vezes não descobrir a rota em tempo razoável.

Figura 2.12: Predicado calcula_trajeto_intermedios

```
calcula_trajeto_intermedios(Origem,Dest,L,Caminho):-
            profundidade_intermedios(Origem,Dest,L,[Origem],Caminho),
            are_members(L,Caminho).

profundidade_intermedios(Dest, Dest,L, H, C):-
            inverso(H,C).

profundidade_intermedios(Origem, Dest,L, His, C):-
            adjacente(paragem(Origem,_,_,_,_,_,_,Carr,_,_,_),paragem(Prox,_,_,_,_,_,_,Carr,_,_,_)),
            \+ member(Prox, His),
            profundidade_intermedios(Prox,Dest,L,[Prox|His],C).

are_members([],L2).

are_members([H],L2):-member(H,L2).

are_members([H|T],L2):-
            member(H,L2),
            are_members(T,L2).
```

Figura 2.13: Exemplo calcula_trajeto_intermedios - em vez de utilizar a rota mais "normal"para chegar ao destino, faz um desvio e passa pelos pontos intermédios

```
| ?- calcula_trajeto_intermedios(182,181,[89,180],R).
R = [182,595,791,183,170,788,68,1014,709,154,87,687,686,742,741,155,159,734,156
,161,162,172,171,799,609,953,597,261,250,107,89,185,594,180,181] ?
yes
| ?-
```

O **nono** predicado é bastante mais complexo que os outros, pois foi adotada uma estratégia por algoritmo informado. Este foi baseado nos algoritmos discutidos nas aulas, como o algoritmo A* e pesquisa gulosa, sendo que o algoritmo tem em consideração a distância a que o próximo nodo fica do Destino. Como se pode deduzir o objetivo é que retorne o caminho entre uma origem e um destino que percorra menos distância. Para isto foi construído um predicado auxiliar que dado as latitudes e longitudes dos nodos retorna essa distância.

Figura 2.14: Predicado resolve_informado_distancia

```
resolve_informado_distancia(Origem, Dest,Caminho/Custo) :-
            assert(goal(Dest)),
            paragem(Origem,LatO,LonO,_,_,_,_,_,_,_),
            paragem(Dest,LatD,LonD,_,_,_,_,_,_,_),
            dist(LatO,LonO,LatD,LonD, Dist),
            informado_distancia([[Origem]/0/Dist], Dest, InvCaminho/Custo/_),
            retract(goal(Dest)),
            inverso(InvCaminho, Caminho).

informado_distancia(Caminhos, Dest, Caminho) :-
            obtem_melhor_distancia(Caminhos, Caminho),
            Caminho = [Nodo|_]/_/_,goal(Nodo).

informado_distancia(Caminhos, Dest, SolucaoCaminho) :-
            obtem_melhor_distancia(Caminhos, MelhorCaminho),
            seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
            expande_informado_distancia(MelhorCaminho, Dest, ExpCaminhos),
            append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
            informado_distancia(NovoCaminhos, Dest, SolucaoCaminho).

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

obtem_melhor_distancia([Caminho1/Custo1/Dist1,_/Custo2/Dist2|Caminhos], MelhorCaminho) :-
            Dist1 =< Dist2, !,
            obtem_melhor_distancia([Caminho1/Custo1/Dist1|Caminhos], MelhorCaminho).

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

expande_informado_distancia(Caminho, Dest, ExpCaminhos) :-
            findall(NovoCaminho, proximoDist(Caminho, Dest, NovoCaminho), ExpCaminhos).

proximoDist([Nodo|Caminho]/Custo/_, Dest, [ProxNodo,Nodo|Caminho]/NovoCusto/Dist) :-
            adjacente(paragem(Nodo,LatO,LonO,_,_,_,_,_,_,_),paragem(ProxNodo,Lat,Lon,_,_,_,_,_,_,_)),
            paragem(Dest,LatF,LonF,_,_,_,_,_,_,_),
            \+ member(ProxNodo, Caminho),
            dist(LatO,LonO,Lat,Lon,D),
            NovoCusto is Custo + D,
            dist(Lat,Lon,LatF,LonF,Dist).

dist(LatO,LonO,LatD,LonD,Dist):-
            Sqrt is (LatD-LatO)^2 + (LonD-LonO)^2,
            Dist is sqrt(Sqrt).
```

Figura 2.15: Resultado resolve_informado_distancia - retorna em conjunto com o caminho a distância percorrida

```
| ?-
| ?- resolve_informado_distancia(97,1009,R).
R = [97,98,717,716,726,727,722,723,751,779,766,691,765,764,228,805,801,1009]/54
76.7194573894985 ?
yes
```

O **décimo** e último predicado é semelhante ao anterior, pois também usa uma política de algoritmo informado. O algoritmo tem como objetivo retornar o caminho entre a origem e o destino que passe por menos paragens possíveis.

Figura 2.16: Predicado resolve_informado_paragens

```
resolve_informado_paragens(Origem, Dest,Caminho/Custo) :-
            assert(goal(Dest)),
            informado_paragens([[Origem]/0], Dest, InvCaminho/Custo),
            retract(goal(Dest)),
            inverso(InvCaminho, Caminho).

informado_paragens(Caminhos, Dest, Caminho) :-
            obtem_melhor_paragens(Caminhos, Caminho),
            Caminho = [Nodo|_]/_,goal(Nodo).

informado_paragens(Caminhos, Dest, SolucaoCaminho) :-
            obtem_melhor_paragens(Caminhos, MelhorCaminho),
            seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
            expande_informado_paragens(MelhorCaminho, Dest, ExpCaminhos),
            append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
            informado_paragens(NovoCaminhos, Dest, SolucaoCaminho).

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

obtem_melhor_paragens([Caminho1/Custo1,_/Custo2|Caminhos], MelhorCaminho) :-
            Custo1 =< Custo2, !,
            obtem_melhor_paragens([Caminho1/Custo1|Caminhos], MelhorCaminho).

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

expande_informado_paragens(Caminho, Dest, ExpCaminhos) :-
            findall(NovoCaminho, proximoParagem(Caminho, Dest, NovoCaminho), ExpCaminhos).

proximoParagem([Nodo|Caminho]/Custo, Dest, [ProxNodo,Nodo|Caminho]/NovoCusto) :-
            adjacente(paragem(Nodo,_,_,_,_,_,_,_,_,_,_),paragem(ProxNodo,_,_,_,_,_,_,_,_,_,_)),
            paragem(Dest,_,_,_,_,_,_,_,_,_,_),
            \+ member(ProxNodo, Caminho),
            NovoCusto is Custo + 1.
```

Figura 2.17: Resultado resolve_informado_paragens - Comparativamente ao calcula_trajeto, devolveu um caminho muito mais pequeno

```
| ?- calcula_trajeto(185,595,R).
R = [185,594,180,181,593,499,182,595] ?
yes
| ?- resolve_informado_paragens(185,595,R).
R = [185,594,595]/2 ?
```

Há também alguns predicados auxiliares fornecidos pelo professor:

Figura 2.18: Predicados Auxiliares

```
inverso(Xs, Ys):-
  inverso(Xs, [], Ys).

inverso([], Xs, Xs).
inverso([X|Xs],Ys, Zs):-
  inverso(Xs, [X|Ys], Zs).

seleciona(E, [E|Xs], Xs).
seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
```

Capítulo 3

Conclusão

Em conclusão, penso que consegui de um modo geral criar um sistema de inferência capaz de responder aos problemas propostos, porém tenho noção que definitivamente não é o mais eficiente, até porque por vezes nem consegue encontrar um caminho dentro de períodos de tempo razoàveis.

Reparei definitivamente que dos meus maiores problemas neste projeto foi realmente construir um algoritmo que não tivesse casos que percorre quase que infinitamente várias paragens à procura de um caminho para o destino devido à grande quantidade de informação, mas que se esta fosse mais pequena os meus predicados encontravam muito mais eficazmente caminhos bons para o problema.

Em trabalhos futuros semelhantes a este diria que é muito importante mesmo focar-me neste problema e modelar eficaz e consisamente um algoritmo informado capaz de lidar com qualquer tipo de situação.

Capítulo 4

Referências

4.1 Referências Bibliográficas

[Analide, 2011] ANALIDE, Cesar, Novais, Paulo, Neves, José,
"Sugestões para a Elaboração de Relatórios",
"Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

4.2 Referências Eletrónicas

Algoritmos de Sorting. Disponível em:
http://kti.mff.cuni.cz/ bartak/prolog/sorting.html. Acesso em 2 jun. 2020.