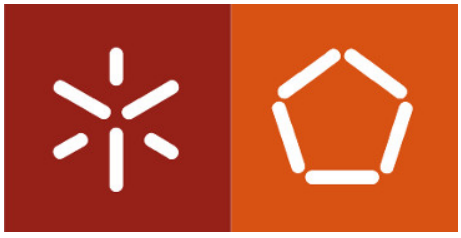
Sistemas de Representação de Conhecimento e Raciocínio

5 de Junho de 2020

a83899 André Morais



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

Este trabalho prático, desenvolvido no âmbito da Unidade Curricular Sistemas de Representação de Conhecimento e Raciocínio, permitiu desenvolver e aplicar os meus conhecimentos em prolog aprendidos ao longo do ano.

O objetivo principal deste trabalho é desenvolver um sistema que permita importar os dados relativos às paragens de autocarro do concelho de Oeiras e representá-los numa base de conhecimento e posteriormente o desenvolvimento de soluções para as queries pedidas.

Toda a implementação do trabalho é explicada nos capítulos seguintes deste relatório.

2 Parser

Como pedido no enunciado do trabalho, tinhamos que fazer parse dos dados dos ficheiros fornecidos em ***.xlsx***. Para mim, foi mais fácil passar,primeiramente, estes ficheiros para um ***.csv*** que foi mais simples de manusear. Criei, por isso, duas classes em ***Java***, uma para cada ficheiro.

2.1 Java

O primeiro ficheiro a que eu fiz parse foi o das paragens de autocarro, obtendo o ficheiro **paragens.pl** onde os dados ficaram definidos

```
paragem(Gid,
        Latitude,
        Longitude,
        "Estado de Conservação",
        "Tipo de Abrigo",
        "Abrigo com Publicidade",
        "Operadora", [Carreira],
        Codigo de Rua,
        "Nome de Rua",
        "Freguesia").
```

Antes de fazer o parse ao segundo ficheiro, foi preciso passar todos as folhas do excel para uma só e só depois passar para ***.csv***. Depois de aplicado o ficheiro java, foi originado o ficheiro **arcos.pl** onde tinha organizado os dados da seguinte maneira (onde a Distancia=$\sqrt{(Lat1 - Lat2)^2 - (Long1 - Long2)^2}$.)

```
arco(Gid1,
     Gid2,
     Distancia,
     Carreira).
```

3 Não Informada - Depth First

A **pesquisa Não Informada** ou **Cega** usa apenas as informações disponíveis na definição do Problema.

Eu optei por escolher a **pesquisa Primeiro em Largura** ou mais conhecida como **Depth First**, em que a estratégia é percorrer todos os nós de menor profundidade primeiramente, só que, normalmente, demora muito tempo e ocupa muito espaço. Em geral só pequenos problemas podem ser resolvidos assim, daí ter optado por este método.

A partir deste tipo de pesquisa, resolvi todos as queries pedidas, havendo em alguns casos ciclos devido a alguma inconsistência/redundância dos dados.

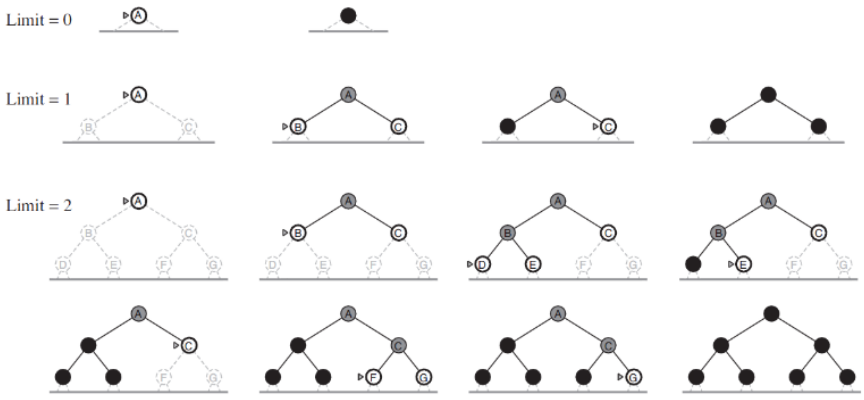


Figura 1: Pesquisa Depth First

3.1 Queries

3.1.1 Query 1

Calcular um trajeto entre dois pontos

Para esta primeira query adaptei o código fornecido pelo o professor Paulo nas aulas práticas da ficha 12, mais concretamente a função, **resolve_pp_c**.

```
[trace]  ?- trajeto(21,11,L,Dist,Cam).
   Call: (8) trajeto(21, 11, _6236, _6238, _6240) ? creep
   Call: (9) trajetoAux(21, 11, _6528, _6238, _6240, []) ? creep
   Call: (10) 21\==11 ? creep
   Exit: (10) 21\==11 ? creep
   Call: (10) adjacente(21, _6532, _6556, _6558) ? creep
   Call: (11) arco(21, _6532, _6556, _6558) ? creep
   Exit: (11) arco(21, 11, 22.28126201858434, 748) ? creep
   Exit: (10) adjacente(21, 11, 22.28126201858434, 748) ? creep
   Call: (10) nao(member(11, [])) ? creep
   Call: (11) lists:member(11, []) ? creep
   Fail: (11) lists:member(11, []) ? creep
   Redo: (10) nao(member(11, [])) ? creep
   Exit: (10) nao(member(11, [])) ? creep
   Call: (10) trajetoAux(11, 11, _6534, _6576, _6578, [21]) ? creep
   Exit: (10) trajetoAux(11, 11, [], 0, [], [21]) ? creep
   Call: (10) _6238 is 22.28126201858434+0 ? creep
   Exit: (10) 22.28126201858434 is 22.28126201858434+0 ? creep
   Call: (10) lists:append([748], [], _6240) ? creep
   Exit: (10) lists:append([748], [], [748]) ? creep
   Exit: (9) trajetoAux(21, 11, [11], 22.28126201858434, [748], []) ? creep
   Exit: (8) trajeto(21, 11, [21, 11], 22.28126201858434, [748]) ? creep
L = [21, 11],
Dist = 22.28126201858434,
Cam = [748] .
```

Figura 2: Exemplo de um output da query 1

Como podemos observar na Figura 4, usei o modo **trace().** para ajudar a perceber como funciona o predicado.

Esta guarda uma lista de visitados, impedindo que este Nodo seja percorrido outra vez e assim impedindo que este entre num loop infinito. Vai percorrendo todos os nodos de modo a chegar ao destino, guardando as carreiras e as distancias , somando a última. Um aspeto importante a apontar nesta implementação é como a lista do caminho é preenchida, pois esta só começa a ser povoada no fim, quando a Origem passa a ser o Destino. Assim este vai passar os elementos da lista dos visitados para a lista do caminho. O mesmo acontece com a lista de Carreiras, é por isso que o caso de paragem é

```
trajetoAux(Destino,Destino, [], 0, [], _).
```

3.1.2 Query 2

Selecionar apenas algumas das operadoras de transporte para um determinado percurso

A estratégia para esta query, foi analoga a da query anterior, onde mudei o predicado adjacente, e dai uma lista de operadoras como argumento de input ao predicado principal. Depois de ir retirar a operadora através da paragem, é so verificar se a operadora dessa paragem pertence a lista de operadoras dadas.

```
?- todosPara(21,23,["Carris"]).
[21,11,650,23],744.4817641624902,[748,201,201]
[21,11,650,23],744.4817641624902,[748,201,748]
[21,11,650,23],744.4817641624902,[748,201,751]
[21,11,650,23],744.4817641624902,[748,748,201]
[21,11,650,23],744.4817641624902,[748,748,748]
[21,11,650,23],744.4817641624902,[748,748,751]
[21,11,650,23],744.4817641624902,[748,751,201]
[21,11,650,23],744.4817641624902,[748,751,748]
[21,11,650,23],744.4817641624902,[748,751,751]
[21,11,650,23],744.4817641624902,[751,201,201]
[21,11,650,23],744.4817641624902,[751,201,748]
[21,11,650,23],744.4817641624902,[751,201,751]
[21,11,650,23],744.4817641624902,[751,748,201]
[21,11,650,23],744.4817641624902,[751,748,748]
[21,11,650,23],744.4817641624902,[751,748,751]
[21,11,650,23],744.4817641624902,[751,751,201]
[21,11,650,23],744.4817641624902,[751,751,748]
[21,11,650,23],744.4817641624902,[751,751,751]
true.
```

Figura 3: Exemplo de um output da query 2

Na figura mostra que do ponto de partida 21 com destino a 23, quem pertencem a operadora Carris, existem 18 caminhos "diferentes"(São todos o mesmo caminho, mas feito em combinações de carreiras diferentes).

3.1.3 Query 3

Excluir um ou mais operadores de transporte para o percurso

A query 3 é muito idêntica à query anterior, mas em vez de verificar se a operadora da paragem pertence, temos de certificar que esta não pertence, fazendo um **nao(member(Operadora1,Operadoras))**.

```
?- todosNaoPara(21,23,["Carris"]).
true.
```

Figura 4: Exemplo de um output da query 3

Neste caso o output vai dar lista vazia, pois a operadora Carris é a única que faz do percurso 21-23.

3.1.4 Query 4

Identificar quais as paragens com o maior número de carreiras num determinado percurso

Nesta query tive que ir a todas as paragens do caminho e fazer o comprimento da lista de carreiras, guardando estas como um par **(Gid,Tamanho)**.

```
?- maiores(21,23).
[(21,2),(11,3),(650,3),(23,3)],[748,201,201]
[(21,2),(11,3),(650,3),(23,3)],[748,201,748]
[(21,2),(11,3),(650,3),(23,3)],[748,201,751]
[(21,2),(11,3),(650,3),(23,3)],[748,748,201]
[(21,2),(11,3),(650,3),(23,3)],[748,748,748]
[(21,2),(11,3),(650,3),(23,3)],[748,748,751]
[(21,2),(11,3),(650,3),(23,3)],[748,751,201]
[(21,2),(11,3),(650,3),(23,3)],[748,751,748]
[(21,2),(11,3),(650,3),(23,3)],[748,751,751]
[(21,2),(11,3),(650,3),(23,3)],[751,201,201]
[(21,2),(11,3),(650,3),(23,3)],[751,201,748]
[(21,2),(11,3),(650,3),(23,3)],[751,201,751]
[(21,2),(11,3),(650,3),(23,3)],[751,748,201]
[(21,2),(11,3),(650,3),(23,3)],[751,748,748]
[(21,2),(11,3),(650,3),(23,3)],[751,748,751]
[(21,2),(11,3),(650,3),(23,3)],[751,751,201]
[(21,2),(11,3),(650,3),(23,3)],[751,751,748]
[(21,2),(11,3),(650,3),(23,3)],[751,751,751]
true.
```

Figura 5: Exemplo de um output da query 4

Como podemos ver aprensenta quantas carreiras tem cada paragem.

3.1.5 Query 5

> Escolher o menor percurso (usando critério menor número de paragens);

Na query 5, aproveitei o resultado do predicado da query 1 que me dava os caminhos todos e apliquei-lhe 3 novos predicados, o **tamanho**, o **filter** e o **filter**. O tamanho devolve (Tamanho da Lista, Lista). O filter imprime o minimo de todas as listas. E por fim aplico o filter para dar filter ao menor número

```
?- menorPercurso(460,486).
(3,[460,468,486]),[1,13]
(3,[460,468,486]),[13,13]
true .
```

Figura 6: Exemplo de um output da query 5

3.1.6 Query 6

> Escolher o percurso mais rápido (usando critério da distância)

Na query 6, apliquei o mesmo pensamento, aproveitando a solução da query 1 e depois aplicando lhe predicados, manipulando da maneira que eu queria apresentar os resultados

```
?- maisRapido(460,486).
[460,468,486],212.00289660864925
[460,468,486],212.00289660864925
true .
```

Figura 7: Exemplo de um output da query 6

3.1.7 Query 7

Escolher o percurso que passe apenas por abrigos com publicidade

A query 7 é parecida à query das Operadoras, mas em vez de ser Operadoras, temos de verificar que se entrada "Abrigo com Publicidade", seja igual a "Yes".

```
?- todosPubli(460,486,L).
[460,468,485,486],238.57945167062576,[1,1,1]
[460,468,485,486],238.57945167062576,[1,1,11]
[460,468,485,486],238.57945167062576,[1,11,1]
[460,468,485,486],238.57945167062576,[1,11,11]
[460,468,486],212.00289660864925,[1,13]
[460,468,485,486],238.57945167062576,[13,1,1]
[460,468,485,486],238.57945167062576,[13,1,11]
[460,468,485,486],238.57945167062576,[13,11,1]
[460,468,485,486],238.57945167062576,[13,11,11]
[460,468,486],212.00289660864925,[13,13]
```

Figura 8: Exemplo de um output da query 7

3.1.8 Query 8

Escolher o percurso que passe apenas por paragens abrigadas

Nesta query, tratei os dados como os anteriores, verificando se o tipo de Abrigo era diferente de "Sem Abrigo"e adicionar a uma lista de abrigos, para saber qual foi o tipo

```
?- todosAbrigo(460,486,L).
[460,468,485,486],238.57945167062576,[1,1,1],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[1,1,11],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[1,11,1],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[1,11,11],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,486],212.00289660864925,[1,13],[Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[13,1,1],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[13,1,11],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[13,11,1],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,485,486],238.57945167062576,[13,11,11],[Fechado dos Lados,Fechado dos Lados,Fechado dos Lados]
[460,468,486],212.00289660864925,[13,13],[Fechado dos Lados,Fechado dos Lados]
```

Figura 9: Exemplo de um output da query 8

3.1.9 Query 9

> Escolher um ou mais pontos intermédios por onde o percurso deverá passar

Na última query, aproveitando o output da primeira query, eu verifico se as paragens passados ao predicado pertencem ao caminho, se todos perteceram, então devolvo esse caminho, como podemos observar na figura seguinte:

```
?- todosEntre(21,23,[11,650]).
[21,11,650,23],744.4817641624902,[748,201,201]
[21,11,650,23],744.4817641624902,[748,201,748]
[21,11,650,23],744.4817641624902,[748,201,751]
[21,11,650,23],744.4817641624902,[748,748,201]
[21,11,650,23],744.4817641624902,[748,748,748]
[21,11,650,23],744.4817641624902,[748,748,751]
[21,11,650,23],744.4817641624902,[748,751,201]
[21,11,650,23],744.4817641624902,[748,751,748]
[21,11,650,23],744.4817641624902,[748,751,751]
[21,11,650,23],744.4817641624902,[751,201,201]
[21,11,650,23],744.4817641624902,[751,201,748]
[21,11,650,23],744.4817641624902,[751,201,751]
[21,11,650,23],744.4817641624902,[751,748,201]
[21,11,650,23],744.4817641624902,[751,748,748]
[21,11,650,23],744.4817641624902,[751,748,751]
[21,11,650,23],744.4817641624902,[751,751,201]
[21,11,650,23],744.4817641624902,[751,751,748]
[21,11,650,23],744.4817641624902,[751,751,751]
```

Figura 10: Exemplo de um output da query 9

4 Informada - A Estrela

A pesquisa informada utiliza a informação do problema para evitar que o algoritmo de pesquisa fique em loop. Esta pode ser mais vantajosa em termos de otimização. A parte mais importante desta técnica é a função heurística ou no meu caso, o predicado distancia em que calcula a distância da Origem ao Destino de forma direta, ignorando os outros caminhos, sendo assim o mínimo possivel para esta mesma.

4.1 Queries

4.1.1 Query 1 e 6

Os predicados para resolver a A Estrela, foram aproveitados também dos da aula do professor Paulo da ficha 12, alterando sempre que chamavamos o Nodo, passavamos a chamar um par(Origem,Destino). Para além disso, também foi criada a função distância e inverso.

```
?- resolve_aestrela(21,23,C).
C = [(21, 23),  (11, 23),  (650, 23),  (23, 23)]/744.4817641624902
```

Figura 11: Exemplo de um output da query 1 e 6

Como pudemos observar, o output do predicado devolve sempre o par **(Nodo, Destino)** e a **Distância** percorrida percorrida.

5 Comparação

	Pesquisa Informada	Pesquisa Não Informada
Conhecimento	Usa conhecimento para encontrar as etapas para a solução	Não usam
Eficiência	Altamente eficiente	Média
Custo	Baixo	Comparativamente Alto
Otimal	Sim	Se todos os custos forem iguais
Completo	Sim	Não
Algoritmo	A*	Depth First

Figura 12: Tabela de comparação dos dois algoritmos

6 Conclusão

Dado como terminado este trabalho, concluo que este permitiu-me aprofundar os conhecimentos obtidos nas aulas teóricas e práticas, percebendo melhor na prática as diferenças entre a pesquisa Não Informada e a Informada.

Fiquei a perceber que a pesquisa Informada, A* estrela, em comparação, com a Depth First, que é altamente eficiente e tem um custo baixo, sendo tambem Otimal e Completo.

Para concluir, neste trabalho todos os objetivos esperados foram alcançados, classificando a minha performance como satisfatória.

Numa perspetiva futura, podia melhorar a nossa implementação ao nível da A* e resolver aqueles casos que entram em loop.

7 Anexos

7.1 Java que gera paragens.pl

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {

        String fich = "/home/morais/Downloads/srcr/paragem_autocarros_oeiras.csv";
        String line = "";
        String split = ";";

        try (BufferedReader buffer = new BufferedReader(new FileReader(fich))) {

            while ((line = buffer.readLine()) != null) {

                // use comma as separator
                String[] paragem = line.split(split);

                System.out.println("paragem(" + paragem[0]  + "," + paragem[1]  +
                "," + paragem[2]  + "," + "\"" + paragem[3] + "\""  + "," + "\"" +
                paragem[4] + "\""  + "," + "\"" + paragem[5] + "\""  + "," + "\"" +
                paragem[6] + "\""  + "," + "[" + paragem[7] + "]" + "," + paragem[8] +
                "," + "\"" + paragem[9] + "\"" + "," + "\"" + paragem[10] + "\"" + ").");

            }

        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

7.2 Java que gera lista.pl

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class Listas {
    public static void main(String[] args) {
        String fich = "/home/morais/Downloads/srcr/lista.csv";
        String line = "";
        String split = ";";
        List<String> l = new ArrayList<>();
        int i;
        int distancia;
        try (BufferedReader buffer = new BufferedReader(new FileReader(fich))) {
            while ((line = buffer.readLine()) != null) {
                l.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        List<String[]> lista;
        lista = l.stream().map(x -> x.split(",")).collect(Collectors.toList());
        for(i=0;i<1561;i++) {
            System.out.println("arco(" + lista.get(i)[0] + "," + lista.get(i+1)[0] +
            "," + Math.sqrt(Math.pow((Double.parseDouble(lista.get(i)[1]) -
            Double.parseDouble(lista.get(i+1)[1])),2) +
            Math.pow((Double.parseDouble(lista.get(i)[2]) -
            Double.parseDouble(lista.get(i+1)[2])),2)) + ","  + lista.get(i)[7] + ")");
        }
    }

}
```

7.3 Implementação Depth First

```
trajeto(Origem,Destino, [Origem|Caminho], Dist,Carreira) :-
trajetoAux(Origem, Destino,Caminho, Dist,Carreira,[]).

trajetoAux(Destino,Destino, [], 0, [],_).
trajetoAux(Origem,Destino ,[Prox|Caminho], Dist, Carreira,Visitados) :-
    Origem \== Destino,
    adjacente(Origem, Prox, Dist1, Carreira1),
    nao(member(Prox,Visitados)),
    trajetoAux(Prox, Destino, Caminho, Dist2, Carreira2,[Origem|Visitados]),
    Dist is Dist1 + Dist2,
    append([Carreira1],Carreira2, Carreira).

adjacente(Origem, Prox, Dist,Carreira) :-
    arco(Origem,Prox, Dist,Carreira).


todos(Origem,Destino,L):-
    findall((S,Dist,Car),trajeto(Origem,Destino,S,Dist,Car),L),
    escrever(L).
```

7.4 Implementação A*

```
resolve_aestrela(Origem, Destino, Caminho/Custo) :-
    distancia(Origem, Destino, X),
    aestrela([[(Origem, Destino)]/0/X], InvCaminho/Custo/_),
    inverso(InvCaminho, Caminho).

aestrela(Caminhos, Caminho) :-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [(Nodo, Destino)|_]/_/_,
    Nodo == Destino.

aestrela(Caminhos, SolucaoCaminho) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela(MelhorCaminho, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela(NovoCaminhos, SolucaoCaminho).


obtem_melhor([Caminho], Caminho) :- !.
obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
    Custo1 + Est1 =< Custo2 + Est2, !,
    obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
obtem_melhor([_|Caminhos], MelhorCaminho) :-
    obtem_melhor(Caminhos, MelhorCaminho).


expande_aestrela(Caminho, ExpCaminhos) :-
    findall(NovoCaminho, adjacente(Caminho,NovoCaminho), ExpCaminhos).


adjacente([(Nodo,Destino)|Caminho]/Custo/_,[(ProxNodo,Destino),(Nodo,Destino)|Caminho]/NovoC
    arco(Nodo, ProxNodo, PassoCusto,_),
    \+ member(ProxNodo, Caminho),
    NovoCusto is Custo + PassoCusto,
    distancia(ProxNodo, Destino, Est).

distancia(Origem, Destino, X) :-
    getCoord(Origem,LatO,LongO),
    getCoord(Destino,LatD,LongD),
    X is sqrt((LatO-LatD)^2 + (LongO-LongD)^2).

getCoord(Ori,Lat,Long):-
    paragem(Ori,Lat,Long,_,_,_,_,_,_,_,_).
```

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

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