Universidade do Minho Departamento de Informática

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Individual

A84590 José Pinto

Resumo

O presente documento relata o processo de desenvolvimento do trabalho individual da unidade curricular Sistemas de Representação de Conhecimento e Raciocínio, inserida no plano curricular de Engenharia Informática da Universidade do Minho. O sistema aqui documentado, escrito em Prolog, pretende caracterizar o universo do sistema de transportes do concelho de Oeiras.

Índice

Lista de Figuras

Introdução

O projeto desenvolvido visa a caracterização de um sistema capaz de lidar com o cálculo de trajetos entre 2 ou mais pontos com diferentes restrições. Para resolver esta questão, pretendo tirar partido da extensão à programação em lógica, escrito em PROLOG, realizando algoritmos de pesquisa. Características como a capacidade de representar todos os caminhos possíveis, cálculo e comparação de distâncias e travessias em grafos com arcos pesados, são alguns dos fatores determinantes desta base de conhecimentos.

Descrição do Trabalho e Análise de Resultados

Base de conhecimento

De forma a retirar a informações necessárias para a base de conhecimento foi feito um pequeno programa em JAVA responsável ler e fazer o parsing necessário dos ficheiros excel conhecimento. As informações retiradas são escritas para um ficheiro baseConhecimento.pl, que é posteriormente importado pelo ficheiro do projeto.

De modo a que o sistema de representação de conhecimentos seja corretamente caracterizado, foram utilizados os seguintes predicados:

- paragem: # Gid, latitude, longitude, estado, Tipo de Abrigo, Abrigo com Publicidade, Operadora, Carreira, Código de Rua, Nome da Rua, Freguesia -> {V,F}
- aresta: #Gid_origem, Gid_destino -> {V,F}

Com estes predicados foi possível representar com sucesso um grafo com todos os trajetos possíveis, onde as paragens são os nodos, cujo predicado guarda toda a informação relativa a cada paragem e as arestas representam as arestas do grafo.

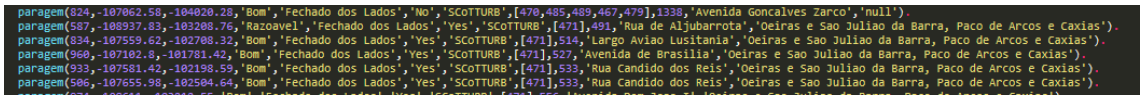

Figura 1- Base de conhecimento excerto das paragens

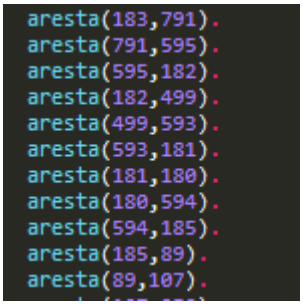

Figura 2- Base de conhecimento excerto das arestas

Algoritmos de Pesquisa

Para a realização das Queries propostas pela equipa docente, foi necessário realizar alguns dos algoritmos de pesquisa falados no âmbito da cadeira de Sistemas de Representação de Conhecimento e Raciocínio. As diferentes estratégias de pesquisa utilizadas são um misto de pesquisa não-informada e informada. Estas permitiram não só a obtenção de resultados, como também a comparação de tempos de execução dos diferentes algoritmos.

Os algoritmos realizados são:

- Caminho(não-informada)

```
%resolução caminho
caminho(A,B,P) :- caminho1(A,[B],P).
caminho1(A,[A|P1],[A|P1]).
caminho1(A,[Y],P):- aresta(X,Y), caminho1(A,[X,Y],P).
caminho1(A,[Y|P1],P) :-
    aresta(X,Y), \+ memberchk(X,[Y|P1]), caminho1(A,[X,Y|P1],P).
```

Figura 3 – algoritmo caminho

- Resolve_pp(não-informada)

```
%resolução pp hist Para nao repetir soluções
resolve_pp(Nodo, Destino, [Nodo|Caminho]) :-
    profundidadeprimeiro(Nodo, Destino, Caminho,8).

profundidadeprimeiro(Nodo, Destino,[],LIMIT) :- LIMIT>0,
    Nodo == Destino, !.

profundidadeprimeiro(Nodo, Destino,[ProxNodo|Caminho],LIMIT) :-
    LIMIT>0, INC is LIMIT - 1, !,
    aresta(Nodo, ProxNodo),
    profundidadeprimeiro(ProxNodo, Destino,Caminho,INC).
```

Figura 4-algoritmo resolve_pp

- Resolve_aestrela(informada)

```
%aestrela
resolve_aestrela(Nodo, Destino, Caminho/Custo) :-
        weight_list([Nodo,Destino],Estima),
        aestrela([[Nodo]/0/Estima], InvCaminho/Custo/_, Destino,10),
        inverso(InvCaminho, Caminho),!.

aestrela(Caminhos, Caminho, Destino,LIMIT) :- LIMIT>0,
        obtem_melhor(Caminhos, Caminho),
        Caminho = [Nodo|_]/_/_,
        Nodo==Destino.

aestrela(Caminhos, SolucaoCaminho, Destino, LIMIT) :-
        LIMIT>0, INC is LIMIT - 1, !,
        obtem_melhor(Caminhos, MelhorCaminho),
        seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
        expande_aestrela(Destino,MelhorCaminho, ExpCaminhos),
        append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
        aestrela(NovoCaminhos, SolucaoCaminho, Destino,INC).


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(Destino, Caminho, ExpCaminhos) :- !,
        findall(NovoCaminho, adjacente(Destino,Caminho,NovoCaminho), ExpCaminhos).

adjacente(Destino,[Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
        aresta(Nodo, ProxNodo),\+ member(ProxNodo, Caminho),
        weight_list([Nodo,ProxNodo],PassoCusto),
        NovoCusto is Custo + PassoCusto,
        weight_list([ProxNodo,Destino],Est).
```

Figura 5-algoritmo aestrela

- Resolve_Gulosa(informada)

```
%gulosa
resolve_gulosa(Nodo, Destino,Caminho/Custo) :-
    weight_list([Nodo,Destino],Estima),
    agulosa([[Nodo]/0/Estima], Destino, InvCaminho/Custo/_,8),
    inverso(InvCaminho, Caminho).

agulosa(Caminhos, Destino, Caminho,LIMIT) :- LIMIT > 0,
    obtem_melhor_g(Caminhos, Caminho),
    Caminho = [Nodo|_]/_/_, Nodo == Destino.

agulosa(Caminhos, Destino,  SolucaoCaminho,LIMIT) :-
    LIMIT>0, INC is LIMIT - 1, !,
    obtem_melhor_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_gulosa(MelhorCaminho, Destino, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
        agulosa(NovoCaminhos, Destino, SolucaoCaminho,INC).


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

obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
    Est1 =< Est2, !, %>
    obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).

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

expande_gulosa(Caminho, Destino, ExpCaminhos) :-
    findall(NovoCaminho, adjacente(Destino, Caminho,NovoCaminho), ExpCaminhos).
```

Figura 6-algoritmo gulosa

A pesquisa não-informada encontra-se representada pelo algoritmo **caminho** e **resolve_pp**. Já a pesquisa informada é representada pelos algoritmos **resolve_aestrela** e **resolve_gulosa**. Sendo que tanto o **resolve_aestrela** como a **resolve_gulosa** têm em consideração as distâncias entre pontos, ao contrário dos outros dois algoritmos. Para todas as Queries que envolvem o cálculo de trajetos entre dois ou mais pontos, foi feita uma versão de resolução para cada um dos algoritmos de pesquisa para posteriormente fazer uma comparação de resultados o mais completa possível.

De forma a evitar possíveis ciclos infinitos optei inicialmente por colocar um limite no número de interações máximas de um algoritmo. A intenção por detrás deste limite estava em tratar soluções fora do tempo útil como resultados negativos de pesquisa. No entanto acabei por retirar este mesmo limite exceto para o algoritmo **resolve_aestrela** uma vez que a inclusão de um limite de interações priva os algoritmos de alcançar alguns caminhos válidos.

Calcular um trajeto entre dois pontos

No enunciado do trabalho um dos pontos pedidos é o cálculo do trajeto entre dois pontos. Para tal foi necessário utilizar os algoritmos de pesquisa referidos anteriormente.

Todos eles são capazes de encontrar qualquer trajeto possível entre dois pontos. A principal diferença está nos tempos de execução que variam consoante o algoritmo de caso para caso.

Mais à frente no relatório encontra-se uma pequena síntese dos diferentes tempos de execução.

```
| ?- caminho(183,594,P).
P = [183,791,595,182,499,593,181,180,594] ?
yes
| ?-
| ?- resolve_pp(183,594,P).
P = [183,791,595,182,499,593,181,180,594] ?
yes
| ?-
| ?- resolve_aestrela(21,23,P).
P = [21,11,650,23]/744.4817641624902 ?
yes
| ?-
| ?- resolve_gulosa(21,23,P).
P = [21,11,650,23]/744.4817641624902 ?
yes
```

Figura 7-Exemplos de percursos

Selecionar apenas algumas das operadoras de transporte

Para esta alínea tirei partido da função caminho acima definida, mas neste caso passa a receber mais um argumento que a anterior. Este argumento trata-se da lista de operadoras que o utilizador escolheu para o seu percurso. Sendo para cada paragem por onde passa o trajeto é feito um **memberchk** da sua operadora na lista das operadoras selecionadas de forma a apenas deixar passar trajetos com paragens pertencentes às operadoras pretendidas.

Para os outros algoritmos de pesquisa foi adicionado uma função auxiliar **verificaOPS** que verifica se as paragens dos trajetos encontrados são válidas.

```
%Selecionar apenas algumas das operadoras de transporte para um determinado percurso
caminhoOP(A,B,OP,P) :- paragem(B,_,_,_,_,_,OPERADORA,_,_,_,_), memberchk(OPERADORA,OP), caminho1OP(A,[B],OP,P).
caminho1OP(A,[A|P1],OP,[A|P1]).
caminho1OP(A,[Y|P1],OP,P) :-
                        aresta(X,Y),
                         \+ memberchk(X,[Y|P1]),
                        paragem(X,_,_,_,_,_,OPERADORA,_,_,_,_),memberchk(OPERADORA,OP),
                         caminho1OP(A,[X,Y|P1],OP,P).

resolve_pp_OP(A,B,OP,P) :- resolve_pp(A,B,P), verificaOPS(P,OP).

resolve_aestrela_OP(A,B,OP,P) :- resolve_aestrela(A,B,P/_), verificaOPS(P,OP).

resolve_gulosa_OP(A,B,OP,P) :- resolve_gulosa(A,B,P/_), verificaOPS(P,OP).

%##############################Funções auxiliares
verificaOPS([],OP).
verificaOPS([X|T],OP) :- paragem(X,_,_,_,_,_,OPERADORA,_,_,_,_), memberchk(OPERADORA,OP), verificaOPS(T,OP).
```

Figura 8 – selecionar operadora

Excluir um ou mais operadores de transporte

O raciocínio para esta alínea foi muito semelhante ao da alínea anterior, no entanto para a função caminho em vez de de **memberchk**, utilizo **\+ memberchk**. Desta forma só são permitidas paragens que não pertencem à lista selecionada.

Para os outros algoritmos de pesquisa foi adicionado uma função auxiliar **verificaN** que verifica se as paragens dos trajetos encontrados são válidas.

```
%Excluir 1 ou mais operadores de transporte para o percurso
caminhoOP_R(A,B,OP,P) :- paragem(B,_,_,_,_,_,OPERADORA,_,_,_,_), \+ memberchk(OPERADORA,OP), caminho1OP_R(A,[B],OP,P).
caminho1OP_R(A,[A|P1],OP,[A|P1]).
caminho1OP_R(A,[Y|P1],OP,P) :-
                        aresta(X,Y),
                         \+ memberchk(X,[Y|P1]),
                        paragem(X,_,_,_,_,_,OPERADORA,_,_,_,_), \+ memberchk(OPERADORA,OP),
                        caminho1OP_R(A,[X,Y|P1],OP,P).


resolve_pp_OP_R(A,B,OP,P) :- resolve_pp(A,B,P), verificaN(P,OP).

resolve_aestrela_OP_R(A,B,OP,P) :- resolve_aestrela(A,B,P/_), verificaN(P,OP).

resolve_gulosa_OP_R(A,B,OP,P) :- resolve_gulosa(A,B,P/_), verificaN(P,OP).


%############################auxiliares
verificaN([],OP).
verificaN([X|T],OP) :- paragem(X,_,_,_,_,_,OPERADORA,_,_,_,_), \+memberchk(OPERADORA,OP), verificaN(T,OP).
```

Figura 9- excluir operadoras

Paragens com o maior número de carreiras num determinado percurso

Nesta alínea comecei por calcular apenas uma das paragens com o valor máximo de paragens para um determinado percurso. A função responsável por tal é a **maisCarreiras** que testa se o cabeça de lista é aquele que tem o maior número de carreiras e caso não seja avanço na lista que representa o trajeto.

Após uma nova leitura mais atenta do enunciado apercebi-me que não era o pretendido. Resolvi então adaptar o que tinha feito anteriormente de forma a em vez de devolver apenas uma das paragens com o número máximo de paragens devolver uma lista com todas as paragens do trajeto que têm esse número máximo de carreiras.

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

%Calcula a lista de carreiras com o maior número de carreiras
carreiras_MaxCarreiras(P,C) :- maisCarreiras(P,M), matchCarreiras(M,P,C).

matchCarreiras(M,[],C).
matchCarreiras(M,[X|T], C) :- maiorNrCarreiras(X,M), matchCarreiras(M, T, C1), adicionar(C1,X,C).
matchCarreiras(M,[X|T], C) :- nao(maiorNrCarreiras(X,M)), matchCarreiras(M, T, C).

%calcula uma das carrerias com o valor máximo de carreiras
maisCarreiras([X],X).
maisCarreiras([X|XS],X) :- maisCarreirasAux(XS,X).
maisCarreiras([X|XS],R) :- nao(maisCarreirasAux(XS,X)), maisCarreiras(XS,R).

maisCarreirasAux([],R).
maisCarreirasAux([X|XS], R) :- nao(maiorNrCarreiras(X,R)), maisCarreirasAux(XS,R).

maiorNrCarreiras(ID1,ID2):-  paragem(ID1,_,_,_,_,_,_,C,_,_,_), paragem(ID2,_,_,_,_,_,_,CR,_,_,_) ,length(C,T1), length(CR,T2), T1 >= T2.
```

Figura 10 – maior número de carreiras

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

Para esta alínea foi feita uma variante para cada um dos algoritmos de pesquisa.

Todas estas versões seguem o mesmo raciocínio, que passa por calcular todos os trajetos possíveis e depois passar a lista de trajetos à função **menorDaLista** responsável por calcular o trajeto com menor número de paragens.

```
%Escolher o menor percurso usando critério menor número de paragens;
%resolução caminho
menorNumeroParagens_CAM(A,B,P) :- findall(T,caminho(A,B,T),L), menorDaLista(L,P).

%resolução pp
menorNumeroParagens_PP(A,B,P) :- findall(T,resolve_pp(A,B,T),L), menorDaLista(L,P).

%resolução aestrela
menorNumeroParagens_Ast(A,B,P) :- findall(T,resolve_aestrela(A,B,T/_),L), menorDaLista(L,P).

%resolução gulosa
menorNumeroParagens_Gul(A,B,P) :- findall(T, resolve_gulosa(A,B,T/_),L), menorDaLista(L,P).

%#########auxiliares
menorDaLista([X],X).
menorDaLista([X,Y|T], M) :- length(X,N1), length(Y,N2), N1<N2, menorDaLista([X|T],M).
menorDaLista([X,Y|T], M) :- length(X,N1), length(Y,N2), N1 >= N2,menorDaLista([Y|T],M).
```

Figura 11– menor número de paragens

Escolher o percurso mais rápido (usando critério da distância)

Nesta alínea existe novamente uma versão para cada um dos algoritmos de pesquisa.

```
%vai buscar todos os caminhos possíveis
%resolução caminho
fastest_path_CAM(A,B,P):- findall(L,caminho(A,B,L),CAMS), check_weights(CAMS,P).

%resolução pp
fastest_path_PP(A,B,P):- findall(L,resolve_pp(A,B,L),CAMS), check_weights(CAMS,P).

%resolução aestrela
fastest_path_Ast(A,B,P):- findall(L,resolve_aestrela(A,B,L/_),CAMS), check_weights(CAMS,P).

%resolução gulosa
fastest_path_Gul(A,B,P):- findall(L,resolve_gulosa(A,B,L/_),CAMS), check_weights(CAMS,P).
```

Figura 12– fastest_path

O que esta função faz é usar os algoritmos de pesquisa para calcular todos os trajetos possíveis e enviar a lista de trajetos para a função **check_weights**.

```
%#########auxiliares
%vai comparando o tamnho das listas até ficar com a menor
check_weights([X],X).
check_weights([X,Y|T],P) :- weight_list(X,D1), weight_list(Y,D2), D1 <  D2, check_weights([X|T],P).
check_weights([X,Y|T],P) :- weight_list(X,D1), weight_list(Y,D2), D1 >= D2,  check_weights([Y|T],P).

%calcula o tamnho de uma lista
weight_list([X],0).
weight_list([X,Y|T],P) :- paragem(X,LAT1,LONG1,_,_,_,_,_,_,_), paragem(Y,LAT2,LONG2,_,_,_,_,_,_,_),
                          dist(LAT1,LONG1,LAT2,LONG2,D), weight_list([Y|T], P1), P = D + P1.

%função responsável por calcular a distância
dist(Lat1,Long1,Lat2,Long2,D):- D is (sqrt((Lat1*Lat2)^2 + (Long1*Long2)^2)).
```

Figura 13- check_weights

Nesta função são comparadas as distâncias totais de cada um dos trajetos. Por sua vez o cálculo de um determinado trajeto é calculado na função **weight_list** que percorre a lista do trajeto acumulando as distâncias que calcula através da função **dist**.

A fórmula usada em **dist** para calcular a distância entre dois pontos é:

$$\sqrt{(Latitude1 - Latitude2)^2 + (Longitude1 - Longitude2)^2}$$

Escolher o percurso que passe apenas por abrigos com publicidade

Aqui foram novamente adicionadas uma restrição aos caminhos encontrados, sendo que agora são apenas escolhidas paragens que têm “YES” no campo da publicidade.

```
% Escolher o percurso que passe apenas por abrigos com publicidade;
caminhoPub(A,B,P) :- paragem(B,_,_,_,_,'Yes',_,_,_,_,_), caminho1Pub(A,[B],P).
caminho1Pub(A,[A|P1],[A|P1]).
caminho1Pub(A,[Y|P1],P) :-
    aresta(X,Y), \+ memberchk(X,[Y|P1]), paragem(X,_,_,_,_,'Yes',_,_,_,_,_),caminho1Pub(A,[X,Y|P1],P).


resolve_pp_PUB(A,B,P) :- resolve_pp(A,B,P), verificaPUB(P).

resolve_aestrela_PUB(A,B,P) :- resolve_aestrela(A,B,P/_), verificaPUB(P).

resolve_gulosa_PUB(A,B,P) :- resolve_gulosa(A,B,P/_), verificaPUB(P).

%#########auxiliares
verificaPUB([]).
verificaPUB([X|T]) :- paragem(X,_,_,_,_,'Yes',_,_,_,_,_), verificaPUB(T).
```

Figura 14-publicidade nas paragens

Escolher o percurso que passe apenas por paragens abrigadas

Mais uma vez foi adicionada uma restrição aos caminhos encontrados pelos algoritmos de pesquisa, mas desta vez, o que é verificado é o campo que identifica o tipo abrigo da paragem. Nesse campo só são validadas strings diferentes de “Sem Abrigo”. Desta forma são apenas escolhidas paragens com algum tipo de abrigo.

```
% Escolher o percurso que passe apenas por paragens abrigadas;
caminhoAbrig(A,B,P) :- paragem(B,_,_,_,ABG,_,_,_,_,_,_), ABG \= 'Sem Abrigo', caminho1Abrig(A,[B],P).
caminho1Abrig(A,[A|P1],[A|P1]).
caminho1Abrig(A,[Y|P1],P) :-
    aresta(X,Y), \+ memberchk(X,[Y|P1]), paragem(X,_,_,_,ABG,_,_,_,_,_,_), ABG \= 'Sem Abrigo', caminho1Abrig(A,[X,Y|P1],P).


resolve_pp_Abrig(A,B,P) :- resolve_pp(A,B,P), verificaAbrig(P).

resolve_aestrela_Abrig(A,B,P) :- resolve_aestrela(A,B,P/_), verificaAbrig(P).

resolve_gulosa_Abrig(A,B,P) :- resolve_gulosa(A,B,P/_), verificaAbrig(P).

%############auxiliares
verificaAbrig([]).
verificaAbrig([X|T]) :- paragem(X,_,_,_,ABG,_,_,_,_,_,_), ABG \= 'Sem Abrigo', verificaAbrig(T).
```

Figura 15-paragens com abrigo

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

Para esta alínea é passado uma lista de pontos intermédios para além do ponto origem e ponto destino. Depois de calculado o percurso é verificado se todos os pontos intermédios se encontram na lista percurso. Se algum dos pontos intermédios não for verificado o caminho é rejeitado.

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

%resolve caminho
caminhoComParagens_CAM(A,B,INT,CAM):- caminho(A,B,CAM), passaIntermedios(INT,CAM).

%resolução pp
caminhoComParagens_PP(A,B,INT,CAM):- resolve_pp(A,B,CAM), passaIntermedios(INT,CAM).

%resolução aestrela
caminhoComParagens_Ast(A,B,INT,CAM):- resolve_aestrela(A,B,CAM/_), passaIntermedios(INT,CAM).

%resolução gulosa
caminhoComParagens_Gul(A,B,INT,CAM):- resolve_gulosa(A,B,CAM/_), passaIntermedios(INT,CAM).

%########auxiliares
passaIntermedios([],CAM).
passaIntermedios([X|T],CAM) :- memberchk(X,CAM), passaIntermedios(T,CAM).
```

Figura 16 – paragens intermédias

Análise de Resultados

De seguida encontra-se uma pequena síntese dos tempos de execução para os diferentes algoritmos. Onde se representa apenas se a resposta é encontrada ou não em tempo útil para percursos longos.

	Trajeto na mesma carreira	**Saltos entre carreiras**	***Findall* Mesma Carreira**	***Findall* Saltos entre carreiras**	**Trajetos impossíveis**
Caminho	Tempo útil	Tempo útil	Infinito	Infinito	Infinito
Resolve_pp	Tempo útil	Tempo útil	Tempo útil	Tempo útil	Infinito
Resolve_aestrela	Infinito	Infinito	Infinito	Infinito	Infinito
Resolve_Gulosa	Tempo útil	Tempo útil	Tempo útil	Tempo útil	Infinito

Convém ainda referir que as operações de **findall** sobre os algoritmos **resolve_aestrela** e **resolve_gulosa**, dão sempre a penas um caminho válido.

Conclusões e Sugestões

No final deste exercício, conclui que foi desenvolvido com sucesso um sistema de caraterização universo do sistema de transportes do concelho de Oeiras. Onde é possível responder a todas as Queries propostas pela equipa docente.

No entanto a minha solução podia cobrir um maior leque de algoritmos de pesquisa, bem como poderia ter sido implementado um histórico nos algoritmos de pesquisa de forma a não repetirem caminhos. Para além disso a própria interface do programa podia ser mais elaborada.

Na conclusão do projeto sou capaz de afirmar um aumento no conhecimento e compreensão do funcionamento dos algoritmos de pesquisa, bem como a aplicação destes à resolução de problemas. A mesma evolução no conhecimento se verificou relativamente à programação lógica estendida e à ferramenta Prolog.

Referências

- Russell and Norvig (2009). Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594.
- Ivan Bratko (2000), PROLOG: Programming for Artificial Intelligence, 3rd Edition, AddisonWesley Longman Publishing Co., Inc.