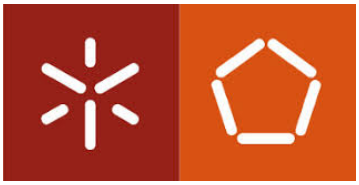

Universidade do Minho
Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Individual

Braga
31 de maio de 2021

Carlos Ferreira
(A89542)

Resumo

O presente relatório faz referência ao trabalho prático individual da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio da Universidade do Minho.

Este trabalho faz uso da extensão à programação em lógica usando a linguagem de programação PROLOG para otimizar, e resolver computacionalmente um problema lógico relacionado a grafos e distâncias eficientemente. Seguindo diversas metodologias de resolução pretende-se descobrir a melhor, mais eficiente e segura escolha perante um problema.

Ao longo do relatório serão então apresentadas os diversos processos de pensamento adotados e desenvolvidos assim como as suas características e funcionamento, pretendendo-se no fim observar os resultados e apresentar criticamente as diferentes metodologias mostradas.

Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio foi proposto, utilizando a linguagem de programação em lógica PROLOG, desenvolver a analisar diferentes algoritmos com a capacidade de otimizar e posteriormente recomendar uma solução que fosse a mais eficiente e rigorosa perante um cenário envolvendo grafos distancias e custos.

Este cenário, resume-se a existência de um camião de lixo que deve, partindo da sua garagem, percorrer diversas ruas e recuperar em cada uma delas os contentores de lixo até assim alcançar a sua capacidade máxima, depositar esse lixo e voltar para a sua garagem.

As ruas têm uma localização e estão obviamente distantes entre si, assim como a quantidade de lixo e conteúdo em cada uma delas não é igual, pelo qual deve ser apresentada diversas soluções dependendo dos critérios a avaliar, como a menor distância percorrida, numero de ruas visitadas,mais lixo recolhido, entre outros.

Para isso foi necessário numa primeira fase construir um parser que lê-se toda a base de conhecimento e a transforma-se em entidades que possam ser posteriormente analisada pelo Prolog, foi então nesta fase usada a linguagem Java. Posteriormente foi necessário construir os algoritmos em Prolog. Ambas as fases serão apresentadas no relatório, o código será demonstrado e explicado assim como os resultados observados e criticados.

Capítulo 2

1ª Fase : Parser

Após o ficheiro com a informação das localizações dos contentores que foi oferecido no formato .xlsx ser convertido em formado .txt a construção de um Parser em Java capaz de transformar essa informação em entidade que o Prolog possa utilizar foi bastante simples. A decisão que foi necessário ser feito foi exatamente escolher qual o tipo de entidades que devem ser criadas e quais as suas características acabando-se por escolher duas entidades distintas:

- contentor: #IdContentor,#IdRua, Tipo de Lixo,Capacidade,Quantidade,Volume→ {*V, F*}
- rua: #IdRua, Coordenada X, CoordenadaY →{*V, F*}

Como podemos ver uma rua pode possuir mais do que um contentor sendo preciso a criação de pelo menos 2 entidades, esta informação é toda a que foi necessário para a resolução dos problemas propostos.

Para perceber melhor o resultado obtido daremos o seguinte exemplo:

- Input:-9,143308809;38,70807879;355;Misericórdia;15805: R do Alecrim (Par→(26→30): R Ferragial - R Ataíde);Lixos;CV0090;90;1;90
- Output:contentor(355,15805,'Lixos',90,1,90),rua(15805,-9.143308809,38.70807879).

O output total de fazer parsing de toda a informação está presente então num ficheiro base_conhecimento.pl que não vai ser mostrado por ser muito espaçoso.

A maneira como o Parser é realizado pode ser observado no excerto abaixo, mas resumidamente a cada linha do ficheiro é feito a separação pelo carácter ";"fazendo print imediatamente do contentor presente naquela linha e guardando a identificação e coordenadas da rua num Map No final todas as ruas previamente lidas são então apresentadas. É possível também formar bases de conhecimentos de contentores com somente um tipo de lixo.

```
import java.util.HashMap;
import java.util.Map;

public class main {

    public static void main (String arg[]) throws IOException {
        File file = new File("datasetUTF8.txt");
        BufferedReader br = new BufferedReader(new FileReader(file));
        String tipo = "Lixos";

        Map<String,String> ruasx = new HashMap<>();
        Map<String,String> ruasy = new HashMap<>();

        String st;
        while ((st = br.readLine()) != null) {
            var as = st.split(";", 10);
            if (tipo == "todos"|| as[5].equals(tipo))
            {
                    var rua = as[4].split(":",2);
                    String posX=as[0].replace(",", ".");
                    String posY=as[1].replace(",", ".");
                    ruasx.put(rua[0],posX);
                    ruasy.put(rua[0],posY);
                    System.out.print("contentor("+as[2]+","+rua[0]+",'"+as[5]+"',"
                    +as[7]+","+as[8]+","+as[9]+").\n");
                }
        }
        for (Map.Entry<String,String> entry:ruasx.entrySet()){
                String nome=entry.getKey();
                String x = entry.getValue();
                String y = ruasy.get(entry.getKey());
                System.out.println("rua("+nome+","+x+","+y+").");
        }
    }
}
```

Capítulo 3

2ª Fase : Realização de algoritmos de Otimização

3.1 Dados Iniciais

Observando a quantidade enorme de dados na nossa base de conhecimento, acompanhado da existência da capacidade máxima da stack do prolog, infelizmente, é bastante complicado para computadores normais resolver este tipo de problemas, para tal, a base de dados foi diminuida um pouco e consequentemente a carga do camião foi também reduzida.

Numa fase Inicial é necessário indicar qual a carga máxima do camião e também adicionar duas novas ruas, a rua referente a localização da garagem e a rua referente a localização do depósito, devemos usar ids não existentes na criação destas novas ruas.

Adicionalmente, foram utilizados algumas flags, para fazer include da nossa base de conhecimento, impedir a redução da escrita no ecrã, e também recusar um certo tipo de avisos.

```
1 :- include('base_conhecimento.pl').
2 :-set_prolog_flag(answer_write_options,[max_depth(0)]).
3 :-style_check(-discontiguous).
4
5 rua(21940,-9.143308809,38.70807879).
6 rua(21941,-9.143308809,38.70807879).
7
8 cargaMax(3000).
```

3.2 Funções auxiliares

Este trabalho necessitou do desenvolvimento de funções auxiliares dos quais algumas muito importantes como verificar se duas ruas são adjacentes, calcular a distância entre duas ruas ou simplesmente retirar um elemento de uma lista, no entanto, deparando-se com dificuldade computacional na resolução de grande quantidade de dados algumas destas funções foram aplicadas de forma simples, mas de forma lógica e que possam ser facilmente alterada no futuro.

```
1  nao( Questao ) :-
2     Questao, !, fail.
3  nao(_).
4
5  membro(X, [X|_]).
6  membro(X, [_|Xs]):-membro(X, Xs).
7
8  inverso(Xs, Ys):-inverso(Xs, [], Ys).
9  inverso([], Xs, Xs).
10 inverso([X|Xs],Ys, Zs):-inverso(Xs, [X|Ys], Zs).
11
12 writeLista([L]):-writeln(L).
13 writeLista([L|T]):-write(L),!,writeLista(T).
14
15 seleciona(E, [E|Xs], Xs).
16 seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
17
18 adicionarLista(X,Q,[],[(X,Q)]):-!.
19 adicionarLista(X,Q,[(X,Y)|T],[(X,O)|T]):- O is Y+Q,!.
20 adicionarLista(X,Q,[(X1,Y)|T],[(X1,Y)|Lista]):- adicionarLista(X,Q,T,Lista),!.
21
22 adjacente(IdRua,IdRuaNovo):-
23         rua(IdRuaNovo,_,_),
24         (R1 is IdRua-IdRuaNovo,R2 is abs(R1),R2<50).
25
26 percorre(IdRua,IdRuaNovo,Dist,DistNovo):-
27         rua(IdRua,XAntes,YAntes),
28         rua(IdRuaNovo,XNovo,YNovo),
29         PerX is XAntes-XNovo,
30         PerY is YAntes-YNovo,
31         DistNovo is Dist + abs(PerX)+abs(PerY).
```

3.3 Estados e Transições

Numa primeira fase, quando pretendemos realizar algoritmos de otimização em linguagem lógica é muito importante aplicar os conceitos de estado e transição. Um estado é toda a informação referente do nosso sistema durante uma iteração, foi decidido guardar bastante informação no nosso estado para que possa ser facilmente utilizada posteriormente nas diversas queries.

Um estado é então representado por:

- Rua: responsável por guardar o id da rua onde o camião se encontra.
- Carga: responsável por avisar qual a carga que o camião tem presente.
- (Tipo,Quantidade): uma lista responsável por guardar quanto lixo de cada tipo o camião tem presente.
- Distancia:responsável por guardar a distância percorrida por um camião.
- Moves: responsável por guardar quantas ruas o camião percorreu.

Quanto as transições, neste projeto, foi decidido, ter somente um tipo chamado mover(IdRua), que indica o processo pelo qual o camião se desloca para uma nova rua.

Para a realização da otimização é necessário indicar qual o estado inicial e qual o estado final. No estado inicial o camião encontra-se na rua da garagem, e todas as outras variáveis são vazias ou zero. O sistema encontra-se no estado final quando a carga do camião é máxima.

- inicial(estado(21940,0,[],0,0)).
- final(estado(_,Carga,_,_,_)):-cargaMax(Carga).

No excerto de código abaixo será apresentado o predicado transição responsável por dando um estado e uma transição calcular o estado resultante de aplicar a transição ao estado dado.

```
carrega(IdContentor,Carga,CargaDepois,Tipo,TipoNovo):-
        contentor(IdContentor,_,TipoAdd,_,Quant,Volume),
        cargaMax(CargaMax),
        CargaDepois is min(CargaMax,Carga+Volume),
        adicionarLista(TipoAdd,Quant,Tipo,TipoNovo).

carregaLista([],C,C,T,T,D,D,_).
carregaLista([Id|Resto],C,C2,T,T2,D,D2,Rua):-
        carrega(Id,C,CD,T,TD),
        (
                (cargaMax(M),CD=:=M,percorre(Rua,21941,D,D3),
                        percorre(21941,21940,D3,D2),T2=TD,C2=CD,!);
                (carregaLista(Resto,CD,C2,TD,T2,D,D2,Rua))
        ).

transicao(estado(IdRua,Carga,Tipo,Dist,Moves),mover(IdRuaNovo),
    estado(IdRuaNovo,CargaDepois,TipoNovo,DistNovoTotal,MovesDepois)):-
        percorre(IdRua,IdRuaNovo,Dist,DistNovo),
        MovesDepois is Moves+1,
        findall(IdContentor,contentor(IdContentor,IdRuaNovo,_,_,_,_),L),
        carregaLista(L,Carga,CargaDepois,Tipo,TipoNovo,
            DistNovo,DistNovoTotal,IdRuaNovo).
```

Como podemos analisar o ato de mover para uma nova rua, implica atualizar a variável distancia, o número de ruas visitadas e após buscar a lista de todos os contentores na rua, carregar todo o seu lixo, vale a pena mencionar que caso, durante o carregamento dos contentores se o camião ficar cheio, faz-se imediatamente o percurso até o depósito e a garagem sendo a variável Distancia atualizada de acordo.

3.4 Tipos de Procura

3.4.1 Procura em Profundidade

A procura em profundidade resume-se em calcular pelo menos uma situação onde o problema é resolvido com sucesso, ou seja, atinge-se o caso final, no entanto, é possível usando o findall e em consequência o backtracking do PROLOG, para calcular todas as possibilidades onde o estado final é alcançado, apesar de ser uma procura muito dispendiosa em tempo, é certamente a mais rigorosa, visto que como temos a totalidade das soluções podemos facilmente escolher com certeza aquela que nos melhor agrada.

```
1  resolved([mover(21940)|Solucao],EstadoFinal):-
2          inicial(InicialEstado),
3          resolvedf(InicialEstado,[mover(21940)],Solucao,EstadoFinal).
4
5  resolvedf(Estado,_,[],Estado):-
6          final(Estado),!.
7
8  resolvedf(Estado,His,[],Estado):-
9          length(His,N),N>10,!,fail.
10
11 resolvedf(Estado,Historico,[mover(IdNovo)|Solucao],EstadoFinal):-
12         Estado=estado(IdRua,_,_,_,_),
13         adjacente(IdRua,IdNovo),
14         nao(member(mover(IdNovo),Historico)),
15         transicao(Estado,mover(IdNovo),EstadoNovo),
16         resolvedf(EstadoNovo,[mover(IdNovo)|Historico],Solucao,EstadoFinal).
```

Como podemos ver, foi decidido usar a procura em profundidade limitada, ou seja, impedir a procura a partir de um certo numero de ruas visitadas, o seu funcionamento apesar de computacionalmente complexo, como podemos ver o excerto código é bastante fácil de desenvolver, resumidamente, a procura começa no estado inicial e em cada iteração se alcançar o estado final dá sucesso e acaba, se alcançar uma certa profundidade dá falhanço, se não for nenhuma destas 2 situações, o sistema ira procurar uma rua adjacente, verificar se o camião já passou por essa rua, e senão fazer a transição para essa mesma rua.

Querys de Profundidade

Após ter a lista com todos os resultados encontrados pela procura em profundidade é necessário, aplicar um filtro a essa lista, encontrando a melhor solução, foram feitos bastantes filtros e as soluções obtidas podem depois ser usadas para verificar a eficiência dos outros algoritmos visto que temos a certeza que estas soluções são as melhores para o seu respetivo critério de avaliação.

```
%Query que devolve a lista de todas as possibilidades encontradas com profundidade
todos(L):-findall((S,C),resolved(S,C),L).

%Query que devolve o numero de possibilidades existente.
mostraPossibilidades():-todos(L),length(L,T),writeln(T).

%Query que calcula o percurso com mais ruas visitadas
calculaMaisParagens(Estado,Resultado):-todos(L),maisParagens(L,Estado,Resultado).

maisParagens([(P,X)],(P,X),R):-length(P,R).
maisParagens([(Px,X),(Py,_)|T],(Pm,M),R):- length(Px,TX),length(Py,TY),TX>TY,
    maisParagens([(Px,X)|T],(Pm,M),R).
maisParagens([(Px,_),(Py,Y)|T],(Pm,M),R):- length(Px,TX),length(Py,TY),TX=<TY,
    maisParagens([(Py,Y)|T],(Pm,M),R).

%Query que calcula o percurso com menos ruas visitadas
calculaMenosParagens(Estado,Resultado):-todos(L),menosParagens(L,Estado,Resultado).

menosParagens([(P,X)],(P,X),R):-length(P,R).
menosParagens([(Px,X),(Py,_)|T],(Pm,M),R):- length(Px,TX),length(Py,TY),TX<TY,
    menosParagens([(Px,X)|T],(Pm,M),R).
menosParagens([(Px,_),(Py,Y)|T],(Pm,M),R):- length(Px,TX),length(Py,TY),TX>=TY,
    menosParagens([(Py,Y)|T],(Pm,M),R).

%Query que calcula o percurso com menor distancia percorrida
calculaMenosDist(Estado,Resultado):-todos(L),menosDist(L,Estado,Resultado).

dist(estado(_,_,_,D,_),D).
menosDist([(P,X)],(P,X),R):-dist(X,R).
menosDist([(Px,X),(_,Y)|T],(Pm,M),R):- dist(X,TX),dist(Y,TY),TX<TY,
    menosDist([(Px,X)|T],(Pm,M),R).
menosDist([(_,X),(Py,Y)|T],(Pm,M),R):- dist(X,TX),dist(Y,TY),TX>=TY,
    menosDist([(Py,Y)|T],(Pm,M),R).

```

```
35 calculaMaisQuant(Estado,Resultado):-todos(L),maisQuant(L,Estado,Resultado).
36
37 %Query que calcula o percurso com mais lixo recolhido de todos os tipos
38 quant(estado(_,_,[],_,_),0).
39 quant(estado(_,_,[(_,B)|T],_,_),D):-quant(estado(_,_,T,_,_),R),!,D is R+B.
40 maisQuant([(P,X)],(P,X),R):-quant(X,R).
41 maisQuant([(Px,X),(_,Y)|T],(Pm,M),R):- quant(X,TX),quant(Y,TY),TX>TY,
42     maisQuant([(Px,X)|T],(Pm,M),R).
43 maisQuant([(_,X),(Py,Y)|T],(Pm,M),R):- quant(X,TX),quant(Y,TY),TX=<TY,
44     maisQuant([(Py,Y)|T],(Pm,M),R).
45
46 %Query que calcula o percurso com mais lixo recolhido de um certo tipo
47 calculaMaisQuantTipo(Tipo,Estado,Resultado):-todos(L),
48     maisQuantTipo(Tipo,L,Estado,Resultado).
49
50 quantTipo(_,estado(_,_,[],_,_),0).
51 quantTipo(X,estado(_,_,[(X,B)|T],_,_),D):-
52     quantTipo(X,estado(_,_,T,_,_),R),!,D is R+B.
53 quantTipo(X,estado(_,_,[(_,_)|T],_,_),D):-
54     quantTipo(X,estado(_,_,T,_,_),D),!.
55
56 maisQuantTipo(Tipo,[(P,X)],(P,X),R):-quantTipo(Tipo,X,R).
57 maisQuantTipo(Tipo,[(Px,X),(_,Y)|T],(Pm,M),R):- quantTipo(Tipo,X,TX),
58     quantTipo(Tipo,Y,TY),TX>TY,maisQuantTipo(Tipo,[(Px,X)|T],(Pm,M),R).
59 maisQuantTipo(Tipo,[(_,X),(Py,Y)|T],(Pm,M),R):- quantTipo(Tipo,X,TX),
60     quantTipo(Tipo,Y,TY),TX=<TY,maisQuantTipo(Tipo,[(Py,Y)|T],(Pm,M),R).
```

3.4.2 Procura em Largura

A procura em largura gasta bastante mais espaço que a profundidade, o que não é completamente bom porque é bastante fácil para o Prolog alcançar o limite da stack, esta procura permito-nos encontrar sempre soluções com menor número de iterações, ou seja, nodos do grafo percorridos, de forma muito rápida e eficaz, infelizmente, por exemplo, menos ruas visitadas não corresponde a menor distância percorrida, é uma procura limitada em resultados, mas muito eficiente para outro tipo de problemas.

Ao contrário da procura em profundidade uma característica muito importante e relevante da procura em largura é que não trabalha somente com um estado, como podemos ver, em cada iteração desta procura irá ser mantido uma lista de estados, e em cada interação cada um desses estados será expandido para todos os estados que possam ser obtidos aplicando uma única transição, e assim sucessivamente até encontrar uma solução nessa lista de estados.

```
1  resolveBF(Solucao,EstadoFinal) :-
2          inicial(InicialEstado),
3          resolveBF([(InicialEstado,[mover(21940)])],Solucao,EstadoFinal).
4
5  checkEstadoFinal([],_,_):-!,fail.
6  checkEstadoFinal([(Es,S)|_],S,Es):-final(Es),!.
7  checkEstadoFinal([_|T],S,Es):-checkEstadoFinal(T,S,Es).
8
9  resolveBF(Estados,Solucao,Estado) :-
10         checkEstadoFinal(Estados,Estado,Solucao).
11
12 expandeLista([],[]).
13 expandeLista([(Estado,[mover(IdRua)|T])|Xs],L):-
14         findall( (Estado1,[mover(IdNovo),mover(IdRua)|T]),
15                  (
16                  adjacente(IdRua,IdNovo),
17                  nao(member(mover(IdNovo),[mover(IdRua)|T])),
18                  transicao(Estado,mover(IdNovo),Estado1)
19                         ),Ls),
20         expandeLista(Xs,L2),
21         append(Ls,L2,L).
22
23 resolveBF([(Estado,T)| Xs], Solucao,EstadoFinal) :-
24         expandeLista([(Estado,T)|Xs],L),
25         resolveBF(L,Solucao,EstadoFinal).
```

3.4.3 Procura Gulosa e Estrela

As procuras Gulosa e Estrela também foram procuras dadas nas aulas e o seu funcionamento é semelhante à procura em largura, mas diverge do facto de não expandir todos os caminhos, mas expandir só aquele que mais favorece o nosso critério. A diferença entre procura gulosa e estrela é que a gulosa ao expandir um caminho ignora completamente os outros que julgou ser menos desejados, enquanto que a estrela guarda esses caminhos.

```
1  resolveGulosa(EstadoFinal, Solucao) :-
2          inicial(EstadoInicial),
3      agulosa([(EstadoInicial,[mover(21940)])],Solucao,EstadoFinal).
4
5  agulosa(Caminhos,Solucao,EstadoFinal) :-
6      obtem_melhor(Caminhos,(EstadoFinal,Solucao)),
7      final(EstadoFinal).
8
9  agulosa(Caminhos, SolucaoCaminho,EstadoFinal) :-
10     obtem_melhor(Caminhos, MelhorCaminho),
11     expande(MelhorCaminho, ExpCaminhos),
12     agulosa(ExpCaminhos, SolucaoCaminho,EstadoFinal).
13
14 resolveEstrela(EstadoFinal, Solucao) :-
15         inicial(EstadoInicial),
16     estrela([(EstadoInicial,[mover(21940)])],Solucao,EstadoFinal).
17
18 estrela(Caminhos,Solucao,EstadoFinal) :-
19     obtem_melhor(Caminhos,(EstadoFinal,Solucao)),final(EstadoFinal).
20
21 estrela(Estados, Solucao,EstadoFinal) :-
22     obtem_melhor(Estados, MelhorEstado),
23     seleciona(MelhorEstado,Estados, OutrosEstados),
24     expande(MelhorEstado, ExpEstados),
25     append(OutrosEstados, ExpEstados, NovoEstados),
26     estrela(NovoEstados,Solucao,EstadoFinal).
27
28 expande((Estado,[mover(IdRua)|T]), ExpCaminho) :-
29         findall( (Estado1,[mover(IdNovo),mover(IdRua)|T]),
30                         (adjacente(IdRua,IdNovo),
31                         nao(member(mover(IdNovo),[mover(IdRua)|T])),
32                         transicao(Estado,mover(IdNovo),Estado1)
33                 ),ExpCaminho).
```

Predicado obtem melhor

Durante a procura gulosa e estrela é necessário encontrar a partir de uma lista de estado aquele que desejamos expandir, para tal, o predicado obtem_melhor é bastante relevante e é a partir dele que vai ser possível escolher o nosso critério de preferência, é importante salientar que na procura em estrela alguns erros possam ser cometidos, por exemplo, escolher o estado com maior carga equivale a fazer procura gulosa, pois não tem em atenção o número de ruas percorridas.

```
obtem_melhor([EstadoFinal],EstadoFinal) :- !.

%obtem melhor com volume/RuasVisitadas
%obtem_melhor( [(estado(A,Carga,B,D,E),Historico),
%   (estado(_,Carga2,_,_,_),Historico2)|Estados], EstadoFinal) :-
%        length(Historico,N1),length(Historico2,N2),
%        R1 is div(Carga,N1),R2 is div(Carga2,N2),
%        R1 >= R2, !,
%        obtem_melhor([(estado(A,Carga,B,D,E),Historico)|Estados], EstadoFinal).

%obtem melhor com dist/RuasVisitadas
%obtem_melhor( [(estado(A,C,B,Dist1,E),Historico),
%   (estado(_,_,_,Dist2,_),Historico2)|Estados], EstadoFinal) :-
%        length(Historico,N1),length(Historico2,N2),
%        R1 is Dist1/N1,R2 is Dist2/N2,
%        R2 >= R1, !,
%        obtem_melhor([(estado(A,C,B,Dist1,E),Historico)|Estados], EstadoFinal).


%obtem melhor com dist menor
obtem_melhor( [(estado(A,C,B,Dist1,E),Historico),
    (estado(_,_,_,Dist2,_),_)|Estados], EstadoFinal) :-
        Dist2 >= Dist1, !,
        obtem_melhor([(estado(A,C,B,Dist1,E),Historico)|Estados], EstadoFinal).

obtem_melhor([_|Estados], EstadoFinal) :- !,
        obtem_melhor(Estados,EstadoFinal).
```

Em todos estes predicados um que deve ser salientado é aquele resultante de escolher o estado com menor distância percorrida, principalmente se usado na procura em estrela, isto porque resulta num algoritmo de procura muito interessante e que nos permite com uma exatidão imensa calcular o percurso com menos distancia percorrida, mostrando muitas vantagens tanto encontradas no algoritmo de procura como em largura.

Capítulo 4

Resultados

Durante os testes realizados, foram encontrados muitos problemas em ajustar parâmetros que não resultassem em tempos de otimização muito grandes, mas principalmente que não alcançassem o limite da stack do PROLOG, foram realizados, por exemplo, testes com tempos de execução na procura em profundidade a chegar aos 2 minutos e mais de um milhão de probabilidade, mas infelizmente durante a execução de outras querys o valor da stack era excedido muito facilmente. Os valores que evitam estes problemas são de facto muito pequenos comparados com a dimensão desejada do problema, no entanto, foi possível fazer uma ótima distinção entre tipo de procuras, analisando rapidez, rigorosidade nos resultados e eficiência em geral. Utilizando outro computador mais apto para estes cálculos seria certo afirmar que os resultados obtidos com este código seria excecionais. Como podemos ver pela imagem abaixo, resultante de executar procuras com o intuito de encontrar caminhos com menor distância percorrida, os resultados foram tal como esperados.

- A procura em largura encontrou o percurso com menos ruas percorridas.
- A procura em profundidade demorou bastante mas alcançou obviamente o resultado ótimo,
- A procura gulosa apresentou um comportamento muito adequado ao seu nome, acabando muito rápido, mas deixando os resultados a expectativa.
- A procura em estrela, tal como esperado, demonstrou ser uma opção maravilhosa, tendo rápidez e conseguindo apresentar o resultado ótimo.

```
% c:/Users/carlo/OneDrive/Ambiente de Trabalho/TPSRCR/TP.pl compiled 0.05 sec, 540 clauses
?- get_time(I),mostraPossibilidades(),get_time(F),Time is F-I.
32274
I = 1622565702.269036,
F = 1622565704.112944,
Time = 1.8439078330993652.

?- get_time(I),calculaMenosDist(E,R),get_time(F),Time is F-I.
I = 1622565737.650723,
E =  ([mover(21940),mover(21941),mover(21944),mover(21927),mover(21889)],estado(21889,3000,[(Vid
ro,3),(Organicos,1),(Lixos,10)],0.011653024000008116,4)),
R = 0.011653024000008116,
F = 1622565739.74511,
Time = 2.0943870544433594 .

?- get_time(I),resolveBF(E,R),get_time(F),Time is F-I.
I = 1622565787.196523,
E = estado(21927,3000,[(Lixos,6),(Embalagens,2),(Vidro,4),(Organicos,3)],0.019285441999997488,2)
,
R = [mover(21927),mover(21959),mover(21940)],
F = 1622565787.19854,
Time = 0.0020170211791992188 .

?- get_time(I),resolveGulosa(E,R),get_time(F),Time is F-I.
I = F, F = 1622565805.448482,
E = estado(21927,3000,[(Vidro,3),(Organicos,3),(Papel e CartÃ£o,2),(Lixos,8),(Embalagens,3)],0.0
30231797999999088,7),
R = [mover(21927),mover(21957),mover(21952),mover(21939),mover(21925),mover(21944),mover(21941),
mover(21940)],
Time = 0.0 .

?- get_time(I),resolveEstrela(E,R),get_time(F),Time is F-I.
I = 1622565813.739884,
E = estado(21889,3000,[(Vidro,3),(Organicos,1),(Lixos,10)],0.011653024000008116,4),
R = [mover(21889),mover(21927),mover(21944),mover(21941),mover(21940)],
F = 1622565814.005169,
Time = 0.26528501510620117 .
```

Capítulo 5

Conclusões

De uma perspetiva de interesse e educação, a realização deste projeto teve bastante sucesso, o uso de linguagem de programação para otimizar problemas e recomendar soluções que, por outro lado, demoraria muito dinheiro e tempo para serem resolvidas de outra forma, é algo que fascina qualquer programador, e que apresenta muitas vantagens para qualquer negócio oferecendo grandes expectativas para o futuro.

Em contrapartida, achei as limitações do computador e do PROLOG bastante problemáticas, impedindo-me de realizar otimizações com enormes quantidades de dados, por outro lado, sabendo que no passado muitos tiveram sucesso com muitas mais limitações, surgiu a lição de estar atento a eficiência do nosso código, tendo em atenção a linguagem e ferramentas utilizadas.

A realização deste trabalho foi bastante esclarecedora dos conteúdos lecionados nas aulas e proporcionou um melhor entendimento das vantagens e desvantagens de linguagens lógicas, e um grande avanço no uso destas mesmas.