Departamento de Informática

UNIVERSIDADE DO MINHO

Métodos de Resolução de Problemas e de Procura em PROLOG

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2º semestre 2020/2021)

82098 Melânia Rafaela Sousa Pereira

Braga,
junho 2021

Resumo

Foi proposto como trabalho individual da Unidade Curricular (UC) Sistemas de Representação do Conhecimento e Raciocínio (SRCR) a realização de um trabalho prático que consiste na criação de um grafo a partir de um *dataset*, e fazer análises com recurso à programação lógica.

Para análise dos dados foi criado um *parser* em Python, que filtra os dados e os coloca numa base lógica de conhecimento, para posteriormente operar sobre eles. Estes dados são provenientes dos circuitos urbanos de Lisboa, da freguesia da Misericórdia.

Para a resolução da procura entre grafos, foram usados três algoritmos de procura: Depth First, pesquisa não informada, A* e Gulosa, pesquisa informada. Todos estes algoritmos foram desenvolvidos e testados em prolog.

Após a execução dos mesmo, obteram-se resultados satisfatórios, sendo possível concluir que os algoritmos de pesquisa informada são mais eficientes do que o algoritmo de pesquisa não informada. Notou-se ainda que os resultados da gulosa e do A* são os mesmos, sendo possível concluir que são algoritmos ótimos. A pesquisa em profundidade apresentou resultados diferentes, um pouco afastados do ótimo, mas igualmente satisfatórios.

Índice

1 Introdução

O presente relatório descreve o desenvolvimento de uma solução para o problema apresentado pelos docentes da unidade curricular Sistemas de Representação de Conhecimento e Raciocínio do terceiro ano do Mestrado Integrado em Engenharia Informática.

Tem como objetivo o desenvolvimento de um sistema de recolha de lixos inteligente numa freguesia da cidade de Lisboa. Para tal é fornecido um *dataset* com todas as informações necessárias, como as ruas existentes e os contentores que delas fazem parte.

Para o desenvolvimento foi necessário ter em mente os conhecimentos adquiridos nas aulas teóricas desta UC sobre os algoritmos de procura tanto informada como não informada e a sua implementação em linguagem PROLOG que foi praticada nas aulas práticas.

Toda a estrutura deste relatório foi baseada nas sugestões presentes no relatório técnico fornecido pelos docentes [ANALIDE et al., 2001].

2 Preliminares

Para o arranque do trabalho prático, para além do enunciado, foi fornecido aos alunos um *dataset*, com diversas informações sobre a rede de resíduos urbanos da freguesia da Misericórdia da cidade de Lisboa.

A informação do ficheiro é relativa ao identificador do contentor, bem como a sua localização, latitude e longitude. Além disso, existe a informação da posição de onde este se encontra na rua, isto é, o nome da rua, seguido das duas ruas que fazem ligação com esta. Há ainda uma coluna referente à capacidade do contentor numa dada localização e o tipo de resíduo a ser recolhido.

2.1 *Parser*

Foi necessário, para o desenvolvimento deste trabalho prático, a realização de um *parser*, que serve para transpor a informação presente no *dataset* para o prolog. Escolheu-se o formato como iria ser feita a base de conhecimento. Optou-se então por fazer 3 predicados:

- `adjacente/3` - que contem como argumentos duas ruas e a distância entre elas;
- `contentores/2` - que contém como argumentos uma rua e uma lista de identificadores de contentores;
- `rua/3` - que contém como argumentos a rua, e as coordenadas (latitude e longitude) dessa rua.

Estes predicados foram divididos em dois ficheiros: `adjacencias.pl` que contém os predicados `adjacente` e `rua` e `contentores.pl` que contém o predicado `rua`.

2.2 Abordagem

Para a realização deste trabalho optou-se por algumas simplificações, uma delas assumir que os camiões de recolha tem capacidade ilimitada, não considerando assim a quantidade recolhida como indicador de produtividade; outra a não consideração dos lados par e ímpar das ruas; e ainda a não consideração de diferentes tipos de lixo, levando à não implementação de uma recolha selecionada.

3 Descrição do Trabalho e Análise de Resultados

Começou por, com o *parser*, se encontrar todas as ruas adjacentes presentes no *dataset*, eliminando adjacências com ruas fora da freguesia Misericórdia. Ainda no *parser*, é também desenvolvido o predicado `contentores`, através da adição dos ids dos contentores a uma lista associada à rua onde se encontram, escrevendo depois esta informação num ficheiro *.pl* com a sintaxe correta. Por fim, percorrendo todas as ruas já lidas do *dataset* é ainda adicionado o predicado `rua` com a latitude e longitude associada a cada uma.

Foi decidido implementar três diferentes algoritmos de pesquisa: um de pesquisa não informada - procura em profundidade primeiro - e dois de pesquisa informada - procura gulosa e A estrela.

3.1 Base de Conhecimento

A base de conhecimento deste trabalho está dividida em dois ficheiros, como já referido, apenas por uma questão de facilidade no *parsing* das informações do *dataset*.

Da base de conhecimento fazem parte as adjacências entre ruas onde estão presentes os contentores e a distância entres elas, ainda os ids de contentores em cada rua e também todas as ruas consideradas nas adjacências associadas à sua latitude e longitude. Para além destes que são desenvolvidos através do *parser*, faz ainda parte da base de conhecimento o predicado `estima`.

3.1.1 Adjacências

As adjacências são feitas de acordo com a informação da coluna PONTO_RECOLHA_LOCAL do *dataset*, onde se pode encontras a rua onde o contentor em causa se encontra e ainda as duas ruas que essa conecta, sendo assim, a rua onde se encontra o contentor é adjacente às outras duas, excepto se não houver nenhum conhecimento sobre elas na freguesia.

Segue um exemplo:

Como podemos ver no *dataset*, a Rua do Alecrim está adjacente à Rua Ferragial e à PC Duque da Terceira. Então, na base de conhecimento existiria uma adjacência entre

a Rua Ferragial e a Rua do Alecrim e também entre a Rua do Alecrim e a PC Duque da Terceira, no entanto apenas existe a segunda pois se assume que a Rua Ferragial não pertence à freguesia visto que não existem coordenadas para ela no *dataset*.

Figura 3.1: Adjacência no *dataset*

```
adjacente(' R do Alecrim ',' R Nova do Carvalho ',314.45905740733394).
adjacente(' R do Alecrim ',' Pc Duque da Terceira ',774.3464641253521).
adjacente(' Lg Corpo Santo ',' R Corpo Santo ',485.014020072145).
adjacente(' Lg Corpo Santo ',' R Bernardino da Costa ',1136.3470501382728).
```

3.1.2 Contentores

O predicado `contentores` serve para saber quais e quantos contentores estão em cada rua, por isso, este predicado contém o nome da rua e uma lista de ids de contentores como se pode ver na seguinte figura.

```
contentores(' R do Alecrim ',[355,356,357,358,359,364,365,366,367,368,369,370,36
↪  0,361,362,363,371,372,373,374,375,376]).
contentores(' Lg Corpo Santo
↪  ',[333,334,335,336,337,338,339,342,343,344,340,341]).
contentores(' R Corpo Santo ',[377,378,379,380]).
contentores(' Tv Corpo Santo ',[381,382,383,384,385]).
contentores(' R Bernardino da Costa
↪  ',[386,387,388,389,390,391,392,393,394,395,396]).
contentores(' R da Boavista ',[418,419,420,421,422,423,424,425,426,427,428,429,4
↪  30,431,432,433,434,435,436,437,438,439]).
contentores(' Bqr do Duro ',[447,448,449,450,451]).
```

3.1.3 Rua

Este predicado existe para ter acesso às coordenadas de cada rua, com o objetivo de posteriormente calcular distâncias entre estas, nomeadamente para o cálculo da estima que se explica na secção seguinte.

Pode ver-se de seguida um excerto deste predicado.

```
rua(' R do Alecrim ',-9.14348180670535,38.7073026157039).
rua(' R Corpo Santo ',-9.14255098678099,38.7073286838222).
rua(' Tv Corpo Santo ',-9.1426225690344,38.7066975168166).
rua(' R Bernardino da Costa ',-9.14305015543156,38.7068559589223).
rua(' Lg Conde-Barão ',-9.15201897924565,38.708606605818).
rua(' Tv Marquês de Sampaio ',-9.14910552718357,38.709073383915).
rua(' R da Boavista ',-9.15079311664937,38.7089055507175).
```

3.1.4 Estima

A estima é um parâmetro necessário para o cálculo dos algoritmos de pesquisa informada. Trata-se da distância em linha reta de um ponto até ao ponto destino, sem necessidade de passar pelos pontos adjacentes. Posteriormente é usado para a execução do A* e da Gulosa.

Para obter este valor, foi usada a seguinte formula matemática , sendo que o ponto inicial é o que está declarado no código.

$$distância\ (A \leftrightarrow B) = \sqrt{(A_X - B_X)^2 + (A_Y - B_Y)^2} \tag{3.1}$$

```
1 final(' Pc São Paulo ').
2
3 estima(Rua,Estima) :- final(X), rua(X,LatI,LongI), rua(Rua,LatF,LongF), Estima
  ↪  is sqrt((LatI-LatF)^2 + (LongI-LongF)^2)*1000.
4
5 ruas(S) :- findall(R,rua(R,Lat,Lon),S).
```

3.2 Procura não informada vs. Procura informada

Procurar algo é uma coisa inerente ao ser humano. Desde os primórdios da humanidade, que é necessária a procura para a sobrevivência. Inicialmente por comida. Atualmente, e após grandes evoluções tecnológicas, as procuras continuam com o seu elevado grau e necessidade, e cada vez mais exigentes.

A nível computacional, é algo que é absolutamente necessário. Tudo no meio digital pode ser descrito como grafos, como seria se não fosse possível fazer procuras sobre estes? Entra aqui a necessidade de evoluir e tornar eficientes as procuras que são feitas. Deixa de ser o famoso método de tentativa/erro, e passam-se a usar métodos que usam, 'inteligência'.

A procura não informada procura de entre o que está a seguir, um caminho que satisfaça o requisito principal de qualquer procura: menos custo. Em contra partida, a procura informada, executa com o mesmo critério, mas tendo 'consciência' que o que está para trás é importante para decidir melhor o que fazer a seguir.

	Procura não informada	**Procura informada**
Eficiência	Altamente eficiente, pouco tempo e recursos	Eficiência é mediadora
Custo	Baixo	Comparativamente alto
Atuação	Encontra a solução mais rapidamente	Mais lenta que a pesquisa informada

3.2.1 Em profundidade primeiro - DFS (Depth First Search)

A pesquisa em profundidade primeiro progride através da expansão do primeiro nodo do grafo de procura, e aprofunda-se cada vez mais, até que o nodo objetivo da procura seja encontrado ou até que encontre um nodo que não tenha nodos adjacentes. Aí a procura retrocede (*backtracking*) e começa no próximo nodo.

Segue-se a implementação do algoritmo na linguagem PROLOG:

```
1  resolvedf([Nodo|Caminho],C,Contentores) :- inicial(Nodo),
   ↪  df(Nodo,[Nodo],Caminho,C,Contentores).
2
3  df(Nodo,_,[],0,[]) :- final(Nodo).
4
5  df(Nodo,Historico,[ProxNodo|Caminho],CustoFinal,Cont) :-
6      adjacente(Nodo,ProxNodo,C),
7      nao(membro(ProxNodo,Historico)),
8      contentores(ProxNodo,ContentoresNovo),
9      df(ProxNodo,[ProxNodo|Historico],Caminho,Custo,Contentores),
10     append(ContentoresNovo,Contentores,Cont),
11     CustoFinal is C+Custo.
```

Foi ainda implementado um predicado que se aplica a todos os caminhos resultantes da pesquisa em profundidade primeiro, e depois compara o custo de cada um para encontrar o caminho com melhor custo.

```
1  todosCusto(R) :- solucoes((S,C),(resolvedf(S,C,Contentores)),R).
2
3  melhorCusto(S,Custo) :- todosCusto(R), menorCusto(R,(S,Custo)).
4
5
6  menorCusto([(P,X)],(P,X)).
7  menorCusto([(Px,X)|L],(Py,Y)) :- menorCusto(L,(Py,Y)), X>Y.
8  menorCusto([(Px,X)|L],(Px,X)) :- menorCusto(L,(Py,Y)), X=<Y.
```

Além disto, foi também implementado um predicado que, da mesma forma que o anterior, encontra o caminho que passa por mais contentores, ou seja, o caminho que faz a maior recolha de lixo.

```
1  todosContentor(R) :- solucoes((S,C,Contentores),(resolvedf(S,C,Contentores)),R).
2
3  melhorRecolha(S,Custo,Contentores) :- todosContentor(R),
   ↪  menorRecolha(R,(S,Custo,Contentores)).
4
5  menorRecolha([(P,X,C)],(P,X,C)).
6  menorRecolha([(Px,X,Cx)|L],(Py,Y,Cy)) :- menorRecolha(L,(Py,Y,Cy)),
   ↪  length(Cx,TamCx), length(Cy,TamCy), TamCx<TamCy.
7  menorRecolha([(Px,X,Cx)|L],(Px,X,Cx)) :- menorRecolha(L,(Py,Y,Cy)),
   ↪  length(Cx,TamCx), length(Cy,TamCy), TamCx>=TamCy.
```

3.2.2 Gulosa (Greedy)

O algoritmo da pesquisa gulosa, é um algoritmo simples e intuitivo, usado na otimização de problemas. Este algoritmo faz uma escolha ótima em cada passo, enquanto tenta encontrar o caminho geral mais ótimo para resolver o problema por inteiro. Este algoritmo é bem sucedido em resolver alguns problemas, como o algoritmo de Dijkstra.

```
1  resolveGulosa(Caminho/Custo) :-
2      inicial(Nodo),
3      estima(Nodo, Estimativa),
4      agulosa([[Nodo]/0/Estimativa], CaminhoInverso/Custo/_),
5      inverso(CaminhoInverso, Caminho).
6
7  agulosa(Caminhos, Caminho) :-
8      obtem_melhor_g(Caminhos, Caminho),
9      Caminho = [Nodo|_]/_/_,final(Nodo).
10
11 agulosa(Caminhos, SolucaoCaminho) :-
12     obtem_melhor_g(Caminhos, MelhorCaminho),
13     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
14     expandeGulosa(MelhorCaminho, ExpCaminhos),
15     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
16     agulosa(NovoCaminhos, SolucaoCaminho).
17
18 obtem_melhor_g([Caminho], Caminho) :- !.
19
20 obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
21     Est1 =< Est2, !,
22     obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
23
24 obtem_melhor_g([_|Caminhos], MelhorCaminho) :-
25     obtem_melhor_g(Caminhos, MelhorCaminho).
26
27 expandeGulosa(Caminho, ExpCaminhos) :-
28     findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).
29
30 adjacenteG([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
31     adjacente(Nodo, ProxNodo, PassoCusto),\+ member(ProxNodo, Caminho),
32     NovoCusto is Custo + PassoCusto,
33     estima(ProxNodo, Est).
34
35 seleciona(E, [E|Xs], Xs).
36 seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
37
38 inverso(Xs, Ys):-
39     inverso(Xs, [], Ys).
40
41 inverso([], Xs, Xs).
42 inverso([X|Xs],Ys, Zs):-
43     inverso(Xs, [X|Ys], Zs).
```

3.2.3 A * (A estrela)

O algoritmo A* é um algoritmo de procura informada, que usa o seu histórico para verificar se está no caminho mais eficiente. É usado o custo total no histórico num determinado vértice, para verificar se o caminho onde está a pesquisar ainda é o mais eficiente de momento. Para este processo, são usados os custos em linha reta, do destino até ao nodo onde o algoritmo se encontra a executar, contrariamente aos algoritmos de pesquisa não informada em que o custo utilizado é até aos nodos adjacentes.

```
1  resolveAEstrela(Caminho/Custo) :-
2      inicial(Nodo),
3      estima(Nodo, Estimativa),
4      aestrela([[Nodo]/0/Estima], CaminhoInverso/Custo/_),
5      inverso(CaminhoInverso, Caminho).
6
7  aestrela(Caminhos, Caminho) :-
8      obtem_melhor(Caminhos, Caminho),
9      Caminho = [Nodo|_]/_/_,final(Nodo).
10
11 aestrela(Caminhos, SolucaoCaminho) :-
12     obtem_melhor(Caminhos, MelhorCaminho),
13     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
14     expandeAEstrela(MelhorCaminho, ExpCaminhos),
15     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
16     aestrela(NovoCaminhos, SolucaoCaminho).
17
18 obtem_melhor([Caminho], Caminho) :- !.
19
20 obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
21     Custo1 + Est1 =< Custo2 + Est2, !,
22     obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
23
24 obtem_melhor([_|Caminhos], MelhorCaminho) :-
25     obtem_melhor(Caminhos, MelhorCaminho).
26
27 expandeAEstrela(Caminho, ExpCaminhos) :-
28 findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).
```

3.3 Análise dos Resultados Obtidos

Apresentam-se de seguida alguns dos resultados obtidos.

```
?- inicial(X).
X = ' R do Alecrim '.

?- final(X).
X = ' Pc São Paulo '.
```

Figura 3.2: Pontos inicial e final

```
?- resolvedf(Caminho,Custo,Contentores).
Caminho = [' R do Alecrim ', ' R Nova do Carvalho ', ' Tv dos Remolares ', ' R São Paulo ', ' Pc São Paulo '],
Custo = 477.2149859487677,
Contentores = [565, 566, 567, 568, 569, 570, 571, 572, 573|...] .
```

Figura 3.3: Resultado do algoritmo de procura em profundidade

```
?- melhorCusto(Caminho,Custo).
Caminho = [' R do Alecrim ', ' Pc Duque da Terceira ', ' R Remolares ', ' Tv Ribeira Nova ', ' Pc São Paulo '],
Custo = 155.90614011791088 .
```

Figura 3.4: Caminho com melhor custo obtido pelo algoritmo de procura em profundidade

```
?- melhorRecolha(Caminho,Custo,Contentores).
Caminho = [' R do Alecrim ', ' R Nova do Carvalho ', ' Tv dos Remolares ', ' Av 24 de Julho ', ' Cais do Sodré ', ' Lg Corpo Santo ', ' R Corpo Santo ', ' Tv Corpo Santo ', ' R Bernardino da Costa '|...],
Custo = 1019.3990268657666,
Contentores = [565, 566, 567, 568, 569, 570, 571, 572, 573|...] .
```

Figura 3.5: Caminho com melhor recolha obtido pelo algoritmo de procura em profundidade

```
?- resolveGulosa(Resultado).
Resultado = [' R do Alecrim ', ' Pc Duque da Terceira ', ' R Remolares ', ' Tv Ribeira Nova ', ' Pc São Paulo ']/155.90614011791078 .
```

Figura 3.6: Resultado do algoritmo de procura gulosa

```
?- resolveAEstrela(Resultado).
Resultado = [' R do Alecrim ', ' Pc Duque da Terceira ', ' R Remolares ', ' Tv Ribeira Nova ', ' Pc São Paulo ']/155.90614011791078 .
```

Figura 3.7: Resultado do algoritmo de procura "A estrela"

4 Conclusões e Sugestões

Este trabalho ajudou bastante na consolidação dos conhecimentos sobre os algoritmos de procura existentes, alguns já estudados em unidades curriculares anteriores, mas outros totalmente novos. Considera-se que algoritmos de procura são uma base fundamental não só na programação e desenvolvimento de software geral como também e cada vez mais no âmbito da inteligência artificial.

Assim, este projeto foi sem dúvida uma mais valia para o conhecimento de algoritmos de procura que podem, no futuro, vir a ser bastante úteis.

Bibliografia

[ANALIDE et al., 2001] ANALIDE, C., NOVAIS, P., & NEVES, J. (2001). *Sugestões para a Elaboração de Relatórios*. Technical report, Departamento de Informática, Universidade do Minho, Portugal.