Universidade do Minho

Programação em Lógica Estendida, Métodos de Resolução de Problemas e de Procura

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

2ºSemestre 2019/20

Author:
Pedro Costa

Docente:
Paulo Novais

5 de Junho de 2020

Conteúdo

1 Introdução

O relatório apresentado diz respeito ao trabalho proposto no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, utilizando uma linguagem de programação em lógica, PROLOG.
Neste projeto, desenvolveu-se métodos de resolução de problemas e de procura aplicados a grafos. O objetivo foi, com base num dataset que contém informação sobre os autocarros de Oeiras, conseguir extrair informação relevante tal como caminhos mais curtos, escolher um mais pontos de intermédios para compor um caminho, entre vários outras opções.

2 Base de conhecimento

Para representar a informação obtida nos datasets dados em forma de grafo, optei pela representação que me parecia mais simples. É obvio que se pode dividir um grafo em nodos e arestas, sendo exatamente isso que fiz. Assim, a minha base de conhecimento tem apenas dois axiomas, sendo estes:

- **Stop**: Marca a existência de uma paragem com um conjunto de atributos, nomeadamente, id, latitude, longitude, estado de conservação, tipo de abrigo, se tem publicidade, operadora, lista de carreiras, código e nome de rua e freguesia onde se situa.
- **Connected**: Indica que existe uma conexão entre duas paragens, indicando ainda qual a carreira pelo qual esta conexão existe e ainda a distância entre as duas paragens.

Para obter a distância entre duas paragens utilizei o cálculo da distância euclidiana.
Para fazer o pré-processamento do dataset e transformá-lo nos axiomas descritos acima, elaborei um parser feito em Java que utiliza a biblioteca **ApachePOI** para fazer parse de excel. Percorri apenas os vários ficheiros, extraindo nodos dum e conexões do outro.
Nota: Considerei que qualquer conexão é bidirecional pois os autocarros conseguem fazer esse percurso em qualquer um dos sentidos.

3 Queries

3.1 Caminho entre dois nodos

Para responder a esta query desenvolvi dois algoritmos diferentes de forma a poder comparar os resultados que obtia.
Um deles foi DFS (Depth-First Search) e o outro foi BFS (Breadth-First Search).
Resultados obtidos para vários cenários:

3.1.1 Trajeto entre 183 e 776

Com BFS obtive a seguinte solução:

```
R = [183,171,799,599,860,861,359,349,643,638,637,361,362,347,86,338,341,342,346
,345,344,363,335,457,458,490,491,56,655,654,59,57,58,62,63,64,61,60,32,33,364,3
30,845,846,333,367,857,856,863,353,354,983,986,977,1002,954,952,823,818,807,710
,792,947,178,693,692,817,813,235,816,236,223,1009,801,805,228,764,765,773,777,7
76] ?
```

Figura 1: Trajeto obtido com BFS entre 183 e 776

Com DFS não foi possível obter uma solução em tempo decente. Isto acontece em vários cenários pois recursividade num grande conjunto de dados com algoritmos não otimizados torna extremamente difícil de obter soluções ocasionalmente.

3.1.2 Trajeto entre 183 e 79

Com BFS obtive a seguinte solução:

```
| ?- bfs(183,79,R).
R = [183,171,799,599,1001,226,232,52,233,231,886,473,470,483,482,476,904,472,90
2,893,465,186,466,467,78,79] ?
```

Figura 2: Trajeto obtido com BFS entre 183 e 79

Com DFS obteve-se um caminho mais longo mas que acontece ao longo duma carreira diretamente:

```
| ?- dfs(183,79,R).
R = [183,791,595,182,499,593,181,180,594,185,89,107,250,261,597,953,609,242,255
,604,628,39,50,599,40,985,608,249,254,622,51,44,251,38,620,45,614,46,42,600,602
,601,48,49,612,613,611,610,336,357,334,339,347,86,85,341,342,365,366,460,468,48
5,486,487,488,469,462,480,494,957,465,186,466,467,78,79] ?
```

Figura 3: Trajeto obtido com DFS entre 183 e 79

3.2 Caminho entre dois nodos com apenas certas operadoras

```
pathOps(Origin,Destiny,Ops,Path) :- pathOpsAux(Origin,[Destiny],Ops,Path).
pathOpsAux(Origin,[Origin|P1],_,[Origin|P1]).
pathOpsAux(Origin,[Y|P1],Ops,Path) :-  connectedBiDir(X,Y,C,_),
stop(X,_,_,_,_,_,_,Operator,_,_,_,_),
stop(Y,_,_,_,_,_,_,OperatorY,_,_,_,_),
member(Operator,Ops),
member(OperatorY,Ops),
       \+ memberchk(X,[Y|P1]),
    pathOpsAux(Origin,[X,(Y,OperatorY,C)|P1],Ops,Path).
```

Trajeto entre 745 e 107 com as operadoras Vimeca e SCoTTURB:

```
| ?- pathOps(745,107,['Vimeca','SCoTTURB'],P).
P = [745,(736,'Vimeca',2),(147,'Vimeca',2),(156,'Vimeca',2),(734,'Vimeca',2),(1
59,'Vimeca',12),(155,'Vimeca',12),(741,'Vimeca',12),(742,'Vimeca',12),(686,'Vim
eca',12),(687,'Vimeca',12),(87,'Vimeca',12),(154,'Vimeca',12),(709,'Vimeca',12)
,(1014,'Vimeca',12),(68,'Vimeca',12),(788,'Vimeca',12),(170,'Vimeca',12),(183,'
Vimeca',12),(791,'Vimeca',1),(595,'Vimeca',1),(182,'SCoTTURB',1),(499,'Vimeca',
1),(593,'Vimeca',1),(181,'Vimeca',1),(180,'Vimeca',1),(594,'Vimeca',1),(185,'SC
oTTURB',1),(89,'Vimeca',1),(107,'Vimeca',1)] ?
yes
```

Figura 4: Trajeto obtido com DFS entre 745 e 107 apenas usando as operadoras vimeca e scotturb

3.3 Caminho entre dois nodos excluindo certas operadoras

```
pathWithoutOperators(Origin,Destiny, Operators, Path) :-
    pathWithoutOperatorsAux(Origin,[Destiny],Operators,Path).
pathWithoutOperatorsAux(Origin,[Origin|P1],_,[Origin|P1]).
pathWithoutOperatorsAux(Origin,[Y|P1],Operators,Path) :- connectedBiDir(X,Y,C,_),
    stop(X,_,_,_,_,_,Operator,_,_,_,_),
    \+ member(Operator,Operators),
       \+ memberchk(X,[Y|P1]),
    pathWithoutOperatorsAux(Origin,[X,Y,(Operator,C)|P1],Operators,Path).
```

Trajeto entre 954 e 58 excluindo a operadora Vimeca.

```
| ?- pathWithoutOperators(954,58,['Vimeca'],P).
P = [954,1002,('LT',114),977,('LT',114),986,('LT',114),983,('LT',114),354,('LT'
,114),353,('LT',114),863,('LT',114),856,('LT',114),857,('LT',114),367,('LT',114
),333,('LT',114),846,('LT',114),845,('LT',114),330,('LT',114),364,('LT',114),33
,('LT',114),32,('LT',114),60,('LT',114),61,('LT',114),64,('LT',114),63,('LT',11
4),62,('LT',114),58,('LT',114)] ?
```

Figura 5: Trajeto obtido com DFS entre 954 e 58 sem usar a operadora Vimeca

3.4 Paragens com maior número de carreiras num determinado percurso

Para responder a esta query percorro o percurso que me é dado e agrupo cada id da lista com o número de paragens correspondentes. De seguida, utilizo uma função auxiliar que mantém apenas os maiores segundos valores duma lista de pares.

```
most_tracks(Path, NTracks) :-
    most_tracks_aux(Path, [], AccNTracks),
    biggest_pairs(AccNTracks, NTracks).
most_tracks_aux([],[P|T1],[P|T1]).
most_tracks_aux([P|T1], Acc, NTracks) :- stop(P,_,_,_,_,_,_,_,Tracks,_,_,_),
             length(Tracks, L),
             most_tracks_aux(T1,[(P,L)|Acc],NTracks).

biggest_pairs(L,R) :- maxList(L,K,R).

maxList([],(_,0),L).
maxList([(P1,P2)|Tail],(M1,M2),[(M1,M2)]) :-
    maxList(Tail,(TM1,TM2),_),
    P2 > TM2,
    M1 is P1,
    M2 is P2.
maxList([(P1,P2)|Tail],(M1,M2),L) :-
    maxList(Tail,(TM1,TM2),L),
    P2 < TM2,
    M1 is TM1,
    M2 is TM2.
maxList([(P1,P2)|Tail],(M1,M2),[H,(P1,P2)|L]) :-
    maxList(Tail,(TM1,TM2),[H|L]),
    P2 == TM2,
    M1 is TM1,
    M2 is TM2.
```

Paragens com maior número de carreiras num percurso entre 183 e 250:

```
| ?- pathClean(183,250,P),most_tracks(P,N).
P = [183,791,595,182,499,593,181,180,594,185,89,107,250],
N = [(595,7),(250,7),(107,7),(185,7),(594,7)] ?
---
```

Figura 6: Trajeto obtido com DFS entre 183 e 250 e paragens com maior número de carreiras

3.5 Percurso com menor número de paragens

Para responder a esta query apenas percorro todos os trajetos obtidos com DFS/BFS e escolho o que têm menos paragens.

```
shortest_in_stops(Origin,Destiny,Path) :-
    bfs(Origin, Destiny, Path),
    \+ (bfs(Origin, Destiny, Path2),
    length(Path, L1),
    length(Path2, L2),
    Path2 \= Path,
    L2 =< L1).
```

Menor número de paragens obtido entre 499 e 78:

```
yes
| ?- dfs(499,78,P).
P = [499,593,181,180,594,185,89,107,250,261,597,953,609,242,255,604,628,39,50,5
99,40,985,608,249,254,622,51,44,251,38,620,45,614,46,42,600,602,601,48,49,612,6
13,611,610,336,357,334,339,347,86,85,341,342,365,366,460,468,485,486,487,488,46
9,462,480,494,957,465,186,466,467,78] ?
yes
| ?- shortest_in_stops(499,78,P).
P = [499,182,595,171,799,599,1001,226,232,52,233,231,886,473,470,483,482,476,90
4,472,902,893,465,186,466,467,78] ?
yes
```

Figura 7: Trajetos obtidos entre 499 e 78 (DFS versus Algoritmo de cima)

3.6 Percurso com menor distância

Para responder a esta query apenas percorro todos os trajetos obtidos com DFS/BFS e escolho o que têm a menor distância total.

```
fastest(Origin,Destiny,MinDist,Path) :-
pathWithDistance(Origin, Destiny, MinDist, Path),
\+ (pathWithDistance(Origin, Destiny, LowerDist, Path2),
Path2 \= Path,
LowerDist =< MinDist).
```

3.7 Percurso que passe apenas por paragens com publicidade

Este exercício foi resolvido com uma implementação adaptada dum algoritmo do estilo de DFS.

```
withAdvertisingOnlyPath(Origin,Destiny, Path) :-
withAdvertisingOnlyPathAux(Origin,[Destiny],Path).
withAdvertisingOnlyPathAux(Origin,[Origin|P1],[Origin|P1]).
withAdvertisingOnlyPathAux(Origin,[Y|P1],Path) :-
connectedBiDir(X,Y,C,_),
  stop(X,_,_,_,_,true,Operator,_,_,_,_),
\+ memberchk(X,[Y|P1]),
  withAdvertisingOnlyPathAux(Origin,[X,Y,(Operator,C)|P1],Path).
```

Abaixo pode-se ver que entre 174 e 274 não existe um caminho que passe apenas por paragens com publicidade. Pode-se ainda ver que entre existe um caminho que passa apenas por paragens com publicidade entre 268 e 378.

```
| ?- withAdvertisingOnlyPath(174,274,P).
no
| ?- withAdvertisingOnlyPath(268,378,P).
P = [268,830,('LT',122),832,('LT',112),827,('LT',112),831,('LT',112),266,('LT',
112),829,('LT',112),282,('LT',122),283,('LT',122),281,('LT',122),294,('LT',122)
,378,('LT',122)] ?
yes
```

3.8 Percurso que passe apenas por paragens abrigadas

Esta query tem exatamente o mesmo estilo que a anterior apenas mudando a condição imposta.

```
shelteredPath(Origin,Destiny, Path) :-
shelteredPathAux(Origin,[Destiny],Path).
shelteredPathAux(Origin,[Origin|P1],[Origin|P1]).
shelteredPathAux(Origin,[Y|P1],Path) :-
connectedBiDir(X,Y,C,_),
stop(X,_,_,_,'Fechado dos Lados',_,Operator,_,_,_,_),
\+ memberchk(X,[Y|P1]),
shelteredPathAux(Origin,[X,Y,(Operator,C)|P1],Path).
```

Trajeto com paragens abrigadas entre 323 e 491:

```
| ?- shelteredPath(323,491,P).
P = [323,351,('Vimeca',2),352,('Vimeca',2),339,('Vimeca',2),347,('Vimeca',1),86
,('Vimeca',1),85,('Vimeca',1),341,('Vimeca',1),342,('Vimeca',1),346,('Vimeca',2
),343,('Vimeca',2),345,('Vimeca',2),344,('Vimeca',2),363,('Vimeca',2),335,('Vim
eca',2),457,('Vimeca',2),458,('Vimeca',2),490,('Vimeca',2),491,('Vimeca',2)] ?
```

Figura 8: Trajeto com paragens abrigadas entre 323 e 491

3.9 Percurso que passa por um ou mais pontos intermédios

Nesta query tirei partido duma função que remove um ponto intermédio da lista que queremos satisfazer e procuro um trajeto até essa lista terminar e se chegar ao nodo de destino.

```
withIntermediate(Origin,Destiny, Intermediate, Path) :-
withIntermediateAux(Origin,[Destiny],Intermediate,Path).
withIntermediateAux(Origin,[Origin|P1],[],[Origin|P1]).
withIntermediateAux(Origin,[Y|P1],Intermediate,Path) :-
    connectedBiDir(X,Y,C,_),
    stop(X,_,_,_,_,_,_,Operator,_,_,_,_,_),
    member(X,Intermediate),
    delete_1(X,Intermediate, New),
    \+ memberchk(X,[Y|P1]),
withIntermediateAux(Origin,[X,Y,(Operator,C)|P1],New,Path).
withIntermediateAux(Origin,[Y|P1],Intermediate,Path) :-
    connectedBiDir(X,Y,C,_),
    stop(X,_,_,_,_,_,_,Operator,_,_,_,_,_),
    \+member(X,Intermediate),
    \+ memberchk(X,[Y|P1]),
    withIntermediateAux(Origin,[X,Y,(Operator,C)|P1],Intermediate,Path).
```

4 Pesquisa informada

4.1 Variação do algoritmo guloso para obter o caminho mais longo

Para obter este algoritmo tirei partimo da seguinte função heuristica que é capaz de determinar entre dois nodos o que encontra mais perto do destino final:

```
estima(Nodo,Destino,Dist) :-
stop(Nodo,LatX,LonX,_,_,_,_,_,_,_,_),
stop(Destino,LatY,LonY,_,_,_,_,_,_,_,_),
Lat is (LatX - LatY) * (LatX - LatY),
Lon is (LonX - LonY) * (LonX - LonY),
Dist is sqrt(Lat+Lon).
```

Assim, torna-se possível escolher a melhor solução dentro do conjunto de nodos ligados ao nodo que estamos a explorar em cada iteração. Tiro partido duma lista com prioridade para ordenar os nodos por interesse, garantindo assim que escolho sempre o caminho mais distante.
No final, removemos os nodos repetidos da lista dos obtidos (devido à existência do mesmo nodo em carreiras diferentes acabamos por ter muitos repetidos a ser tidos em conta desnecessariamente) e invertemos a lista para ficar num sentido de início para fim.
Uma aplicação deste algoritmo podia ser, por exemplo, desenvolver um passeio turístico por Oeiras.
Comparação entre um percurso de 183 a 766 obtido com BFS e por o algoritmo descrito acima:

```
| ?- greedy(183,766,P),distance(P,D).
P = [183,791,595,594,185,107,250,597,953,605,606,609,82,609,599,40,622,51,38,62
0,45,602,601,860,861,359,349,643,638,637,361,362,26,24,22,27,28,641,34,28,86,33
8,342,365,366,460,468,485,486,487,488,469,462,480,494,957,465,186,652,186,466,9
,467,78,79,78,654,59,57,58,62,63,64,61,60,32,33,364,330,845,846,333,367,857,856
,863,353,354,983,986,977,1002,954,952,823,818,807,710,792,947,178,693,692,817,8
13,816,236,223,1009,801,805,808,809,800,229,803,632,804,802,811,810,841,842,837
,835,836,838,454,838,320,324,325,317,319,318,327,326,273,274,396,397,884,437,38
8,387,386,385,389,441,564,551,565,553,552,554,540,10,543,516,503,542,541,869,98
9,379,378,294,281,283,282,829,266,287,828,796,795,191,797,208,785,781,769,768,7
64,765,691,766],
D = 63655.29325493082 ?
yes
| ?- bfs(183,766,P),distance(P,D).
P = [183,171,799,599,860,861,359,349,643,638,637,361,362,347,86,338,341,342,346
,345,344,363,335,457,458,490,491,56,655,654,59,57,58,62,63,64,61,60,32,33,364,3
30,845,846,333,367,857,856,863,353,354,983,986,977,1002,954,952,823,818,807,710
,792,947,178,693,692,817,813,235,816,236,223,1009,801,805,228,764,765,691,766],

D = 26865.588126171468 ?
yes
| ?-
```

Como se pode ver, o percurso obtido pelo greedy é extremamente mais longo que o obtido pelo BFS.

5 Conclusão

Concluindo, foi interessante realizar este trabalho pois permitiu explorar um estilo de programação diferente do que estou habituado.
No entanto, não faço um balanço positivo da exeperiência. Isto deve-se, muito provavelmente, à minha inexperiência com programação em lógica. Digo que faço um balanço negativo uma vez que grande parte dos algoritmos desenvolvidos são incapazes de descobrir uma solução para paragens que não tenham sido escolhidas pre-meditamente.
Acredito, definitivamente, que se tivessem sido desenvolvidos algoritmos mais eficientes o problema acima não se verificaria.