Universidade do Minho
Escola de Engenharia

Universidade do Minho
Departamento de Informática
Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Individual

Gonçalo Soares (A84441)

11 de junho de 2021

Gonçalo Soares

Conteúdo

Introdução

Este relatório tem como objetivo demonstrar o trabalho realizado no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

O objetivo deste trabalho é aplicar os conhecimentos lecionados ao longo deste semestre num caso de estudo.

A equipa docente forneceu-nos um "dataset"em excel, que descreve os pontos de recolha de lixo na cidade de Lisboa, assim será utilizada a linguagem lógica Prolog para criar um conjunto de predicados que respondam a questões relativas ao sistema em questão. Devido à natureza dos dados apresentados, a maneira como estes são melhor representados é através de um grafo em que os nodos são os pontos de recolha e as arestas as ligações entre eles.

Esta resolução limita-se à versão simplificada do problema.

Parser

Todos os dados relativos aos caixotes do lixo e à sua recolha estão disponibilizados no ficheiro dataset.xlsx. Para tratamento dos dados e importação dos mesmos para o prolog, foi utilizada a linguagem de programação python e as bibloteca openpyxl para o parse do ficheiro do tipo .xlsx e a biblioteca re para o uso de expressões regulares. Com o objetivo de deixar todos os pontos de recolha da seguinte maneira: "pontoRecolha(-9.14330880914792,38.7080787857025,355,'Misericórdia',15805,'15805: R do Alecrim (Par (-)(26−30) : $RFerragial-RAtaíde)'$, $[('Lixos', 1860), ('PapeleCartão', 1530)]$).".

```
pontoRecolha(-9.14330880914792,38.7080787857025,355,'Misericórdia',15805,'15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)',[('Lixos', 1860), ('Papel e Cartão', 1530)]).
pontoRecolha(-9.14337777820218,38.7080781891571,364,'Misericórdia',15806,'15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)',[('Lixos', 2760), ('Papel e Cartão', 460)]).
```

Ponto de Recolha

O parser começa por separar toda a informação pertinente e guarda-la em diferentes dicionários. Esta é id da rua, latitude, longitude, lista de lixos por rua,etc. Apesar de também separar as ruas adjacentes, estas não são usadas pois as mesmas arrebemtabam a stack no swiprolog, As ruas adjacentes utilizadas são então as proximas do linhas de cada linha. A baixo o código responsável pelo referido:

```
f = openpyxl.load_workbook(filename='dataset.xlsx', data_only=True)

mDict = {}
ruasAdj = {}
ruasIDName = {}
ruasIDID = {}
lixosID = {}

sheet = f["Folha1"]
for row in sheet.iter_rows():
  a = []
  for cell in row:
    a.append(cell.value)
  mDict[a[2]] = a
  num = re.search(r'\d+',a[4])
  nomeRua = re.search(r': ([-0-9A-zÀ-ú ]*),? ',a[4])
  if num:
    idr = num.group(0)
    if idr in ruasIDName:
      pass
    else:
      ruasIDName[idr] = nomeRua.group(1)
    ruasP = re.search(r':(?:.*): ([,()0-9A-zÀ-ú -]*(?:\([,0-9A-zÀ-ú -]*\))? - [0-9A-zÀ-ú -]*(?:\([,0-9A-zÀ-ú -]*\))?)',a[4])
    if (ruasP and ruasP.group(1)):
      arrayRuasAdj = []
      strs = ruasP.group(1).split(' - ')
      arrayRuasAdj.append(strs[0])
      arrayRuasAdj.append(strs[1])
      ruasAdj[idr] = arrayRuasAdj
for key in ruasAdj:
  strK1 = re.search(r'([0-9A-zÀ-ú -]*) \(',ruasAdj[key][0])
  if strK1 and strK1.group(1):
      strK1 = strK1.group(1)
  else:
    strK1 = ruasAdj[key][0]

  strK2 = re.search(r'([0-9A-zÀ-ú -]*) \(',ruasAdj[key][1])
  if strK2 and strK2.group(1):
      strK2 = strK2.group(1)
  else:
    strK2 = ruasAdj[key][1]

  k1 = get_key(strK1,ruasIDName)
  k2 = get_key(strK2,ruasIDName)

  add = []
  if k1:
    add.append(k1)
  if k2:
    add.append(k2)
  if add != []:
    ruasIDID[key] = add
```

Parse do ficheiro .xlsx

2.1 Base de Conhecimento

Assim que toda a informação está lida e em memória, é hora de a escrever em ficheiro, mas antes, para simplificar o problema, juntamos os lixos da mesma rua em uma lista:

```
f = open("kb.pl", "w")

fstRE = mDict[list(mDict.keys())[1]][4]
fstKey = re.search(r'\d+',fstRE)
idPR = fstKey.group(0)
lastKey = list(mDict.keys())[-1]
lixos = {'Lixos':0,'Papel e Cartão':0,'Embalagens':0,'Vidro':0,'Organicos':0}
for k in mDict:
  if k == 'OBJECTID':
    pass
  else:
    strR = str(mDict[k][4])
    num = re.search(r'\d+', strR)
    if num.group(0) != idPR:
      lixosID[idPR] = lixos
      lixos = {'Lixos':0,'Papel e Cartão':0,'Embalagens':0,'Vidro':0,'Organicos':0}
      lixos[mDict[k][5]] += mDict[k][9]
    else:
      lixos[mDict[k][5]] += mDict[k][9]
      if k == lastKey:
        lixosID[idPR] = lixos
    idPR = num.group(0)
```

Junção dos lixos

```
for k in mDict:
  if k == 'OBJECTID':
    pass
  else:
    strR = str(mDict[k][4])
    num = re.search(r'\d+', strR)
    print
    if num.group(0) != idPR:
      f.write('pontoRecolha(' + str(mDict[k][0]) + ','
      + str(mDict[k][1])+',' +str(mDict[k][2]) + ',\''+
      str(mDict[k][3])+'\','+ str(num.group(0)) +',\'' + str(mDict[k][4]) + '\',')
      auxDict = lixosID[num.group(0)]
      writeLista(f, auxDict)
      f.write(').\n')
    else:
      pass
    idPR = num.group(0)
f.close()
```

Escrita em "kb.pl"

2.2 Grafo

Por fim, depois da base de conhecimento criada, o parser cria o ficheiro "grafo.pl", neletemos todos os vértices e arestas pertencentes ao grafo, a seguinte função escreve o ficheiro "grafo.pl":

```
def writeGrafo(lixosID,ruasIDID):
  f = open("grafo.pl", "w")

  f.write('g2(grafo(')
  lista = list(lixosID.keys())
  for i in range(0, len(lista)):
    lista[i] = int(lista[i])
  f.write(str(lista)+ ')).\n')

  lastKey = list(lixosID.keys())[0]
  lastOne = list(lixosID.keys())[-1]
  #writeAdj(f,lastKey,ruasIDID)
  for key in lixosID:
    if lastKey == key:
      pass
    else:
      if lastOne == key:
        #writeAdj(f,key,ruasIDID)
        f.write('aresta('+lastKey+','+key+').\n')
      else:
        f.write('aresta('+lastKey+','+key+').\n')
        #writeAdj(f,key,ruasIDID)
        lastKey = key
```

Escrita em "grafo.pl"

Predicados

Os predicados desenvolvidos devem ser capazes de cumprir os seguintes requisitos:

- Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território;
- Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher);
- Comparar circuitos de recolha tendo em conta os indicadores de produtividade;
- Escolher o circuito mais rápido (usando o critério da distância);
- Escolher o circuito mais eficiente (usando um critério de eficiência à escolha);

3.1 Profundidade

O predicado dfs/3 implementa Depth-First search com uma lista de nodos visitados.O predicado dfsr/4 funciona como um predicado auxiliar recursivo para calcular o caminho. Quando este é chamado pela primeira vez e chamado com os parâmetros: nodo de origem, nodo de destino, uma lista de visitados com o nodo de origem e o caminho. Sempre que este predicado é visitado, obtém-se o próximo nodo garantindo-se que este é adjacente como predicado adjacente/2, garante-se que o nodo ainda não foi visitado com o predicado member e por fim aplica-se recursivamente o predicado atualizando os valores.

```
%-------------------------------- DfS Depth First Search
dfs(Nodo, Destino, [Nodo|Caminho]):-
    dfsr(Nodo, Destino, [Nodo], Caminho).

dfsr(Nodo, Destino, _, [Destino]):-
    adjacente(Nodo, Destino).

dfsr(Nodo, Destino, Visited, [ProxNodo|Caminho]):-
    adjacente(Nodo, ProxNodo),
    \+ member(ProxNodo, Visited),
    dfsr(ProxNodo, Destino, [Nodo|Visited], Caminho).
```

```
[debug]  ?- dfs(15805,21949,P).
P = [15805, 15806, 15807, 15808, 15809, 15810, 15811, 15812, 15813|...] ▮
```

Exemplo Depth-First search

O mesmo algoritmo é implementado mas tendo em conta quantas vezes certo tipo de lixo é visitado:

```
%----------- dfs

getNumLixo(Nodo, Lixo, R):-
   pontoRecolha(_,_,_,_,Nodo,_,L1),
   getNumLixoAux(L1, Lixo,R).

list_sum([Item], Item).
list_sum([Item1,Item2 | Tail], Total) :-
    list_sum([Item1+Item2|Tail], Total).

getNumLixoAux([], Lixo,0).
getNumLixoAux([(Name,V)|Rest], Lixo,Num) :-
  getNumLixoAux(Rest, Lixo,Num1),
  (Name = Lixo -> Num is Num1 + 1; Num is Num1).

dfs_TL(Nodo, Destino, TipoLixo, [Nodo|Caminho]/Number):-
    getNumLixo(Nodo,Lixo,Acc),
    dfsr_TL(Nodo, Destino, TipoLixo, [Nodo], Caminho/Acc/Number).

dfsr_TL(Nodo, Destino, TipoLixo, _, [Destino]/Acc/Number):-
    adjacente(Nodo, Destino),
    getNumLixo(Destino,TipoLixo,Num),
    Number is Num + Acc.

dfsr_TL(Nodo, Destino, TipoLixo, Visited, [ProxNodo|Caminho]/Acc/Number):-
    adjacente(Nodo, ProxNodo),
    \+ member(ProxNodo, Visited),
    getNumLixo(Destino,TipoLixo,Num),
    NewAcc is Acc + Num,
    dfs_TL(ProxNodo, Destino, TipoLixo, [Nodo|Visited], Caminho/NewAcc/Number).
```

```
[debug]  ?- dfs_TL(15805,15806,'Lixos',P).
P = [15805, 15806]/2 .

[debug]  ?- dfs_TL(15805,15806,'Papel e Cartão',P).
P = [15805, 15806]/2
```

Exemplo tipo de lixo Depth-First search

3.2 Largura

O predicado bfs/3 implementa Breadth-First search com uma lista de nodos visitados.O predicado bfsAux/3 funciona como um predicado auxiliar recursivo para calcular o caminho. Quando este é chamado pela primeira vez e chamado com os parâmetros: nodo de origem, nodo de destino, e a solução. Sempre que este predicado é visitado, obtém-se o próximo nodo garantindo-se que este é adjacente como predicado adjacente/2, garante-se que o nodo ainda não foi visitado com o predicado member e por fim aplica-se recursivamente o predicado atualizando os valores.

```
bfs(Start, End, Solution) :-
  bfsAux([[Start]], End, Solution).


bfsAux([[Node|Path]|_], End, Result) :-
  Node == End, ! , inverso([Node|Path], Result).

bfsAux([Path|Paths], End, Solution) :-
  extend(Path, NewPaths),
  append(Paths, NewPaths, Paths1),
  bfsAux(Paths1, End, Solution).

extend([Node|Path], NewPaths) :-
  findall([NewNode, Node|Path],
          (adjacente(Node, NewNode),
           \+ member(NewNode,[Node|Path]) ),
          NewPaths),!.

extend(Path, []).
```

```
[debug]  ?- bfs(15805,15809,P).
P = [15805, 15806, 15807, 15808, 15809].
```

Exemplo Breadth-First search

Como nos algoritmos da profundidade, este também é adptado para calcular quantas vezes o camião passa por um determinado tipo de lixo:

```
%------------ bfs

resolve_bfs_lixo(Start, End, Lixos, Solution) :-
  bfs_loja([[Start]], End, Lixos, Solution).


bfs_lixo([[Node|Path]|_], End, Lixos, Result) :-
  Node == End, ! , inverso([Node|Path], Result).

bfs_lixo([Path|Paths], End, Lixos, Solution) :-
  extend_loja(Path, NewPaths, Lixos),
  append(Paths, NewPaths, Paths1),
  bfs_lixo(Paths1, End, Lixos, Solution).

extend_lixo([Node|Path], NewPaths, Lixos) :-
  findall([NewNode, Node|Path],
          (adjacente_Lixo(Node, NewNode, Lixos),
           \+ member(NewNode,[Node|Path]) ),
          NewPaths),!.

extend_lixo(Path, [], Lixos).
```

Exemplo tipo de lixo Breadth-First search

3.3 Gulosa

O predicado greedy implementa a pesquisa gulosa. Nesta pesquisa foi utilizado como predicado para calcular o lixo estimado.

```
%-------------------------------- Gulosa

greedy(Nodo, Destino, Caminho/LixoR):-
    estima(Nodo,Destino, E),
    estima(Nodo,V),
    agreedy([[Nodo]/V/E], InvCaminho/LixoR/_, Destino),
    inverso(InvCaminho,Caminho).

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

agreedy(Caminhos, SolucaoCaminho, Destino):-
    get_best_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expand_greedy(MelhorCaminho,ExpCaminhos, Destino),
    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
    agreedy(NovoCaminhos,SolucaoCaminho, Destino).

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

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

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

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

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

estima(P1,R):-
        pontoRecolha(_,_,_,_,P1,_,L1),
        getLixo(L1,R).

estima(P1,P2,R):-
        pontoRecolha(_,_,_,_,P1,_,L1),
        pontoRecolha(_,_,_,_,P2,_,L2),
        append(L1,L2,Lr),
        getLixo(Lr,R).

getLixo([(_, V)],V).
getLixo([(_, V1),(_, V2)|Tail],R):- RR is V1 + V2,
                                    getLixo([(_, RR)|Tail],R).
```

```
[debug]  ?-
|    greedy(15805,21944,P).
P = [15805, 15806, 15807, 15808, 15809, 15810, 15811, 15812|...]/207200
```

Exemplo Greedy

3.4 A* (A estrela)

O predicado aestrela implementa a pesquisa a estrela. Nesta pesquisa foi utilizado como predicado para calcular o lixo estimado.

```
%------------------------------ A*

aestrela(Origin, Goal, Caminho/Custo) :-
    estima(Origin, Goal, Estima),
    estima(Origin, V),
    axestrela([[Origin]/V/Estima], InvCaminho/Custo/_, Goal),
    inverso(InvCaminho, Caminho).

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

axestrela(Caminhos, SolutionCaminho, Goal) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    remove(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela(MelhorCaminho, ExpCaminhos, Goal),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
        axestrela(NovoCaminhos, SolutionCaminho, Goal).

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

```
[debug]  ?- aestrela(15805,21944,P).
P = [15805, 15806, 15807, 15808, 15809, 15810, 15811, 15812|...]/207200
```

Exemplo Greedy

3.5 Outros exemplos

Algoritmos como greedy com outro critério, o número de vezes que um tipo de lixo é recolhido em um determinado percurso:

```
%----------- gulosa

greedyTP(Nodo, Destino,TipoLixo, Caminho/LixoR):-
    estimaTP(Nodo,Destino, TipoLixo,E),
    estimaTP(Nodo,TipoLixo,V),
    agreedyTP([[Nodo]/V/E], TipoLixo,InvCaminho/LixoR/_, Destino),
    inverso(InvCaminho,Caminho).

agreedyTP(Caminhos, TipoLixo,Caminho, Destino):-
    get_best_gTP(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,
    Nodo == Destino.

agreedyTP(Caminhos, TipoLixo,SolucaoCaminho, Destino):-
    get_best_gTP(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expand_greedyTP(MelhorCaminho,TipoLixo,ExpCaminhos, Destino),
    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
    agreedyTP(NovoCaminhos,TipoLixo,SolucaoCaminho, Destino).

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

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

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

expand_greedyTP(Caminho, TipoLixo ,ExpCaminhos, Destino):-
    findall(NovoCaminho, adjacenteTP(Caminho, NovoCaminho, Destino), ExpCaminhos).

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

estimaTP(P1,TipoLixo,R):-
        pontoRecolha(_,_,_,_,P1,_,L1),
        getLixoTP(L1,R).

estimaTP(P1,P2,TipoLixo,R):-
        pontoRecolha(_,_,_,_,P1,_,L1),
        pontoRecolha(_,_,_,_,P2,_,L2),
        append(L1,L2,Lr),
        getLixoTP(Lr,R).

getLixoTP([(_, V)],TipoLixo,V).
getLixoTP([L1,L2|Tail],TipoLixo,R):-
   isLixo(L1,TipoLixo) -> ((isLixo(L2,TipoLixo) -> (getLixoV(L1,V1),getLixoV(L2,V2),RR is V1 + V2, getLixo([(_, RR)|Tail],TipoLixo,R) ;
                                                    getLixoTP(L1|Tail,TipoLixo,R)))
                        ;getLixoTP(L2|Tail,TipoLixo,R)).

isLixo((T,_),TipoLixo) :- T == TipoLixo.

getLixoV((_,V),Z) :- Z is V.
```

Exemplo Greedy

Conclusão

Concluindo , com este trabalho foram aplicados os conhecimentos de Prolog lecionados ao longo deste semestre na unidade curricular de sistemas de representação de conhecimento e raciocino. Nomeadamente conhecimentos relativos à travessia de grafos.

Como trabalho futuro seria relevante implementar os algoritmos para o funcionamento da versão completa deste trabalho prático.

Bibliografia

[Bra00] Ivan Bratko. *PROLOG: Programming for Artificial Intelligence*. 2000.

[Ces11] José Neves Cesar Analide Paulo Novais. "Sugestões para a Redacção de Relatórios Técnicos". Em: *Relatório Técnico, Departamento de Informática, Universidade do Minho* (2011).

[Ces11] [Bra00]