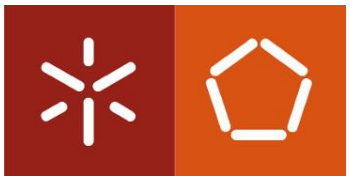


UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

TRABALHO PRÁTICO INDIVIDUAL

Métodos de Resolução de Problemas e de Procura

Sistemas de Representação de Conhecimento e Raciocínio
2º Semestre – 2020/21

Braga, abril de 2021

Raquel Sofia Miranda da Costa – A89464

RESUMO

Ao longo do relatório é apresentada e analisada a solução implementada para o problema proposto pela equipa docente da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

O objetivo deste trabalho prático centra-se na análise de um caso de estudo relacionado com os circuitos de recolha de resíduos urbanos no concelho de Lisboa. Para tal recorre-se à aplicação de diversas estratégias através do uso de algoritmos de procura, além do desenvolvimento de mecanismos de raciocínio adequados a esta problemática. Foram utilizados algoritmos de pesquisa não informada e pesquisa informada, recorrendo aos dados disponíveis, tais como as coordenadas geográficas, no dataset fornecido. Assim sendo, foi fulcral processar o dataset e representar os mesmos dados sob a forma de um grafo.

Além disso, a elaboração da resolução à problemática apresentada deverá permitir, tendo em conta a localização de um ponto de partida, gerar circuitos, comparar e escolher, segundo um conjunto de critérios previamente definidos.

Índice

Índice de Figuras

Introdução

O presente trabalho prático centra-se num caso de estudo referente aos circuitos de recolha de resíduos urbanos no concelho de Lisboa. Para tal é concedido o acesso a um dataset que disponibiliza um conjunto de informações tais como as coordenadas geográficas de cada contentor existente no concelho, as várias ruas, a conexão existente entre as mesmas e ainda o tipo de lixo de cada contentor.

Com base nestas informações é construído um grafo de forma a aplicar os diversos algoritmos de pesquisa informada e não informada. Consequentemente foi fundamental adaptar os mesmos, é importante referir que estes terão diferentes benefícios de acordo com os critérios especificados para o circuito a gerar.

Sendo assim, uma das principais metas deste trabalho é conseguir elaborar um sistema de recomendação de circuitos de recolha para o caso de estudo.

Formulação do Problema

O problema em causa trata-se de um problema de estado único, uma vez que o ambiente é determinístico e totalmente observável, e o agente, neste caso, o camião do lixo "sabe" exatamente o estado em se encontra, e onde é que se encontra o lixo. A solução é, portanto, reduzida à procura de um caminho do estado inicial até ao estado objetivo.

Formulando o problema obtemos os seguintes tópicos:

- **Estado Inicial:** Garagem
- **Estado Objetivo:** Depósito
- **Estados:** Os vários pontos de recolha existentes
- **Operações:** Mover-se entre os vários pontos, e recolher o lixo.
- **Custo da Solução:** É determinado pela distância entre pontos de recolha de cada circuito realizado. Esta reflete uma medida de desempenho da solução.
- **Limitação:** O camião só poderá recolher no máximo $15\ m^3 (= 15\ 000\ L)$ de resíduo. Portanto, assim que ele estiver "cheio" deverá ir ao depósito descarregar e retornar à garagem de modo a reinicializar o circuito de recolha de resíduos.
- **Objetivos:**
  - → Encontrar o circuito mais rápido, utilizando o critério da distância;
  - → Encontrar o circuito mais eficiente, considerando a quantidade de resíduos recolhida e a distância média percorrida. Sendo assim, o circuito que permitir a recolha de uma maior quantidade de resíduo, e uma menor distância percorrida será o mais eficiente.

Carregamento dos dados

Interpretação do "dataset"

Como dataset para o projeto foi fornecido um ficheiro em formato .xlsx, que posteriormente teve de ser convertido em formato .csv, de forma que o prolog conseguisse carregar e interpretar.

Primeiramente foi necessário analisar o dataset dado, de modo a compreender e a distinguir as informações fulcrais à realização do trabalho, daquelas que não são necessárias ou que poderemos dispensar.

Tipicamente em cada linha do ficheiro encontramos as seguintes informações:

- Latitude
- Longitude
- ObjectID
- PontoRecolha_Freguesia
- PontoRecolha_Local
- Contentor_Resíduo
- Contentor_Tipo
- Contentor_Capacidade
- Contentor_QT
- Contentor_Total_Litros

No campo referente ao local do ponto de recolha, existe normalmente um ID desse mesmo ponto, e as ruas às quais esse ponto encontra-se conectado. Por exemplo para o seguinte caso:

$$15805: R\ do\ Alecrim\ (Par\ (->)(26-> 30): R\ Ferragial\ -\ R\ Ataíde)$$

É considerado que poderá ser feita uma deslocação no seguinte sentido R.Ferragial -> R. do Alecrim -> R.Ataíde. Sendo assim, conseguimos obter o sentido de cada aresta do grafo.

Posto isto, considera-se que um ponto de recolha é representado pelas várias linhas do ficheiro Excel que têm o mesmo id relativamente ao campo "PontoRecolha_Local", sendo que cada linha representa um contentor presente nesse mesmo ponto. É ainda importante referir que cada contentor poderá ter um tipo de resíduo associado, tal como por exemplo: "Embalagens", "Vidros", "Papel e Cartão", etc.

Além disso, foi dada também uma especial atenção às coordenadas geográficas de cada ponto, uma vez que, é através delas que é calculada a distância entre dois nodos do grafo. É de realçar que esta foi calculada a partir da Fórmula de *Haversine* que se trata de uma equação usada em navegação, fornecendo distâncias entre dois pontos de uma esfera a partir das latitudes e longitudes dos mesmos.

Base de Conhecimento

Para construir a base de conhecimento recorreu-se ao predicado "csv_read_file" de uma biblioteca do *swi-prolog*. Ao ler cada linha do ficheiro csv, esta mesma é inserida na base de conhecimento sob a forma do predicado **contentor/8**:

- ID do contentor
- Latitude
- Longitude
- ID do ponto de recolha onde se encontra
- Nome da Rua onde se encontra
- Ruas adjacentes
- Tipo de resíduo do contentor
- Capacidade em Litros do contentor

Mas antes de inserir os dados do ficheiro sob a forma de um predicado, foi necessário tratar deles, para tal foi usado o predicado "split_string", de forma a ser possível retirar o ID de um ponto de recolha de uma linha que fosse do tipo:

$$15807: R\ do\ Alecrim\ (Impar\ (5 -> 25)(->): R\ Ferragial\ -\ Pc\ Duque\ da\ Terceira)$$

Assim, sendo foi necessário fazer um split através do caracter ":", tal como ilustrado no seguinte excerto de código:

```
convert_dataset([row(Latitude,Longitude,ObjectId,Ponto_Recolha_Freguesia,Ponto_Recolha_Local,Contentor_Residuo,C
    aux_convert(T).

aux_convert([]).
aux_convert([row(Latitude,Longitude,ObjectId,Ponto_Recolha_Freguesia,Ponto_Recolha_Local,Contentor_Residuo,Conte
    split_string(Ponto_Recolha_Local, ":", " ",[X,Xs|S]),
    createRuasAdjacentes(X,Xs,S,NomeRua,Ruas),
    split_string(NomeRua,"","," ",[NRua|Ns]),
    atom_number(X, Id_Local), %converte de string para int, os outros já estão com o tipo certo
    assert(contentor(ObjectId,Latitude,Longitude,Id_Local,NRua,Ruas,Contentor_Residuo,Contentor_Total_Litros)),
    aux_convert(T).
```

Além disso, de forma a retirarmos as ruas adjacentes em cada ponto de recolha, foi feito o seguinte predicado, que irá usar também o predicado "split_string".

```
% Ver em que rua o ponto de recolha se encontra e quais as ruas adjacentes ao mesmo.
createRuasAdjacentes(X,Xs,[],NomeRua,[]):- split_string(Xs,",","," ",[NomeRua|T]).
createRuasAdjacentes(X,Xs,[S|Ss],NomeRua,Ruas) :- split_string(S,"",")",[H|T]),
                                                   split_string(H,"-"," ",Ruas),
                                                   split_string(Xs,"(","",[NomeRua|Ns]).
createRuasAdjacentes(_,_,[_]).
```

Depois de ser inserido o conhecimento relativo a cada contentor existente no dataset, ou seja, o equivalente a cada linha do ficheiro, foi necessário criar os predicados referentes ao nodo do grafo, às arestas, e ainda às estimativas. Tudo isto é feito no predicado "load_dataset", que está encarregue de converter toda a informação do dataset sob a forma de diversos predicados.

```
load_dataset :- csv_read_file("dataset.csv", Data, [functor(row), arity(10)]),
                convert_dataset(Data),
                create_vertices(),
                %create_adjacenciaInRua(),
                create_AdjacenciaOutRuas(),
                %get_pontosSeq().
                garagem(G), goal(Gl),
                arestas_LocalG(G),arestas_LocalD(Gl),
                create_estimativa(Gl).
```

Tal como vimos, primeiro é feita uma conversão recorrendo a vários *splits*. De seguida, são criados os vértices/nodos do grafo, aos quais são dados o nome de "Ponto de Recolha", através de um *findall,* de modo a criar uma lista de tuplos com o **Id do Ponto**, a **Latitude**, a **Longitude**, **o nome da Rua**, e as **Ruas adjacentes**. Depois de forma a evitar repetições nessa mesma lista é feita uma conversão da lista para uma estrutura do tipo set, através do predicado "list_to_set" do swi-prolog. No final insere-se na base do conhecimento sob a forma do predicado **pontoRecolha/6**.

```
create_vertices():-
findall((LocalId,Latitude,Longitude,Rua,RuasAdj),contentor(_,Latitude,Longitude,LocalId,Rua,RuasAdj,_,_),Locals),
list_to_set(Locals,L),
create_pontoRecolha(L).
```

```
create_pontoRecolha([(Id,Latitude,Longitude,Rua,RuasAdj)|T]) :-
findall(ContentorId,contentor(ContentorId,_,_,Id,_,_,_,_),ListC),
assert(pontoRecolha(Id,Latitude,Longitude,Rua,RuasAdj,ListC)),
create_pontoRecolha(T).
create_pontoRecolha([_]).
```

Em relação às adjacências, considerou-se dois tipos: as adjacências entre ruas diferentes, que é aquela que obtemos ao ler uma linha do ficheiro Excel, e a adjacência entre pontos que se encontram na mesma rua, mas têm diferentes Ids. Este último caso acontece quando existe linhas do ficheiro Excel cujos pontos não contêm nenhuma informação sobre as ruas conectadas ao mesmo. As adjacências são representadas sob a forma do predicado **aresta/3**.

15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)

21961: R do Alecrim, 24

Além disto, foi preciso também escolher dois pontos. Um que representasse a garagem, isto é, de onde o camião começa o seu percurso, e outro o depósito, onde o camião descarrega o lixo que colheu ao longo do circuito efetuado. Para tal foram escolhidos os pontos 15885 e 15889 do dataset, respetivamente. O segundo passo, foi conectar tanto a garagem como o depósito a todos os outros pontos do grafo, considerando sempre que todas as arestas que contêm o ponto da garagem são do tipo: $aresta(Garagem, X, Distância)$ e todos aqueles que contêm o depósito são do tipo: $aresta(X, Depósito, Distância)$. Tal abordagem foi feita de modo a simplificar o problema, porque caso contrário não saberíamos se era possível aceder à garagem ou ao depósito a partir de um determinado ponto do grafo.

É de realçar que o predicado aresta, além de conter o id dos dois nodos que se conectam possui ainda a distância entre ambos, calculada através da fórmula de Haversine.

Foi ainda também calculada a estimativa do custo de cada nodo até ao nodo final, isto é, da distância de um nodo x ao nodo final. Esta informação foi colocada sob a forma do predicado **estima/2**.

Concluindo, no final, obtemos uma base de conhecimento que contêm os seguintes predicados:

```
?- showContentores().
355,-9.14330880914792,38.7080787857025,15805,R do Alecrim,[R Ferragial,R Ataíde],Lixos,90
356,-9.14330880914792,38.7080787857025,15805,R do Alecrim,[R Ferragial,R Ataíde],Lixos,1680
357,-9.14330880914792,38.7080787857025,15805,R do Alecrim,[R Ferragial,R Ataíde],Lixos,90
358,-9.14330880914792,38.7080787857025,15805,R do Alecrim,[R Ferragial,R Ataíde],Papel e Cartão,1440
359,-9.14330880914792,38.7080787857025,15805,R do Alecrim,[R Ferragial,R Ataíde],Papel e Cartão,90
364,-9.14337777820218,38.7080781891571,15806,R do Alecrim,[R Ataíde,R Ferragial],Lixos,240
365,-9.14337777820218,38.7080781891571,15806,R do Alecrim,[R Ataíde,R Ferragial],Lixos,840
366,-9.14337777820218,38.7080781891571,15806,R do Alecrim,[R Ataíde,R Ferragial],Lixos,120
367,-9.14337777820218,38.7080781891571,15806,R do Alecrim,[R Ataíde,R Ferragial],Lixos,180
```

```
?- showPontosRecolha().
15805,-9.14330880914792,38.7080787857025,R do Alecrim,[R Ferragial,R Ataíde],[355,356,357,358,359]
15806,-9.14337777820218,38.7080781891571,R do Alecrim,[R Ataíde,R Ferragial],[364,365,366,367,368,369,370,360,361,362,363]
15807,-9.14348180670535,38.7073026157039,R do Alecrim,[R Ferragial,Pc Duque da Terceira],[371,372,373,374,375,376]
15808,-9.14255098678099,38.7073286838222,R Corpo Santo,[Lg Corpo Santo,Tv Corpo Santo],[377,378,379,380]
15809,-9.14276596081499,38.7070836135523,Tv Corpo Santo,[R Corpo Santo,R Bernardino da Costa],[381,382]
15810,-9.1426225690344,38.7066975168166,Tv Corpo Santo,[R Bernardino da Costa,Cais do Sodré],[383,384,385]
15811,-9.14240835587344,38.7069966274245,R Bernardino da Costa,[Lg Corpo Santo,Tv Corpo Santo],[386,387,388,389,390]
15812,-9.14305015543156,38.7068559589223,R Bernardino da Costa,[Tv Corpo Santo,Pc Duque da Terceira],[391,392,393,394,395,396]
```

```
?- showArestas.
15807,15889,0.06379491792076684
15808,15894,0.05388860574763118
15808,15809,0.035989593954503844
15809,15811,0.04089441293589099
15811,15810,0.04056652316856604
15810,15878,0.0366723778832478
15811,15809,0.04089441293589099
15809,15812,0.04028958729165582
15812,15888,0.04295511925412622
15819,15818,0.17127528614530008
15818,15819,0.17127528614530008
```

```
?- showEstimas.
15805,0.14880847013785134
15806,0.1483905666999655
15807,0.06379491792076684
15808,0.11566338098208315
15809,0.08111272292040961
15810,0.08702969600819271
15811,0.11468206728662274
15812,0.041897881574725515
15813,0.9010023467185767
15814,0.9556017520451596
```

Pesquisa não-informada

Algoritmos de Pesquisa utilizados:

Profundidade (DFS - Depth-First Search)

Para o algoritmo de procura em profundidade foi necessário ter em atenção a quantidade de lixo que o camião levanta em cada ponto de forma a não ultrapassar os 15000L. Além disso, optou-se por adaptar o algoritmo a uma forma generalizada que é capaz de devolver a distância percorrida e a quantidade de lixo levantada no final de um caminho, gerando circuitos de recolha tanto indiferenciada como seletiva.

```
resolve_DFS(Nodo,TipoLixo,Residuo,[Nodo|Caminho],Dist,Qnt):- pp_Alg(Nodo,[Nodo],Caminho,Qnt,Dist,TipoLixo,Residuo,0).


pp_Alg(Nodo,_,[],0,0,TipoLixo,Residuo,_):- goal(Nodo).

pp_Alg(Nodo,Historico,[ProxNodo|Caminho],QntLixo,Dist,TipoLixo,Residuo,Limite):-
                        adjacente(Nodo,ProxNodo,Dist1),
                        nao(membro(ProxNodo, Historico)),
                        getQtLixo(Residuo,TipoLixo,QntLixo1,Nodo),
                        Limite1 is Limite + QntLixo1,
                        Limite1 =< 15000,
                        pp_Alg(ProxNodo,[ProxNodo|Historico],Caminho,QntLixo2,Dist2,TipoLixo,Residuo,Limite1),
                        QntLixo is QntLixo1 + QntLixo2,
                        Dist is Dist1 + Dist2.
```

Largura (BFS - Breadth-First Search)

Em relação ao algoritmo de pesquisa em largura, foi necessário também adaptá-lo à limitação do camião. Este tipo de pesquisa, é caracterizado por fazer uma busca em extensão, ou seja, primeiro começa por expandir o nó da raiz, depois os seus sucessores e assim sucessivamente. Este algoritmo pode ser interpretado como uma “queue” onde os novos sucessores são colocados no final da fila, e o primeiro da fila é selecionado a cada passo.

```
resolve_BFS(Orig,Caminho,Dist,Q):- goal(Dest),
                                   resolve_lp(Dest,[[Orig]],Caminho,Dist,15000).

resolve_lp(Dest,[[Dest|T]|_],Caminho,0,Limite):- reverse([Dest|T],Caminho).

resolve_lp(Dest,[LA|Outros],Caminho,CustoT,Limite):-    Limite > 0,
                                                        LA=[Act|_],
                                                        findall([X|LA],(Dest\==Act,adjacente(Act,X,_),\+ member(X,LA)),Novos),
                                                        adjacente(Act,X,CustoT1),
                                                        getQtLixo('all',_,Q1,Act), Limite1 is Limite-Q1,
                                                        append(Outros,Novos,Todos),
                                                        resolve_lp(Dest,Todos,Caminho,CustoT2,Limite1),
                                                        CustoT is CustoT1 + CustoT2.
```

Busca Iterativa Limitada em Profundidade

Este algoritmo é bastante semelhante ao algoritmo DFS, a única diferença é que estabelecido um limite do número de nós a procurar, ou seja, neste caso um limite na profundidade. Além disso, mais uma vez é aplicado o limite da quantidade de lixo que o camião poderá recolher a cada circuito efetuado.

```
resolve_DFSLimitada(Origem,Caminho,Custo,Limite):- resolve_DFSLimitada2(Origem,[Origem],Caminho,Custo,Limite,15000).

resolve_DFSLimitada2(Origem,_,[],0,_,_):- goal(Origem).

resolve_DFSLimitada2(Origem,Historico,[ProxNodo|Caminho],Custo,Limite,Camiao):-
                                Limite > 0,
                                Camiao > 0,
                                adjacente(Origem,ProxNodo,Custo1),
                                nao(membro(ProxNodo,Historico)),
                                Limite1 is Limite-1,
                                getQtLixo('all',_,Q1,Origem),
                                Camiao1 is Camiao-Q1,
                                resolve_DFSLimitada2(ProxNodo,[ProxNodo|Historico],Caminho,Custo2,Limite1,Camiao1),
                                Custo is Custo1 + Custo2.
```

Pesquisa informada

Algoritmos de Pesquisa utilizados:

Os algoritmos de pesquisa informada utilizam heurísticas de forma a auxiliar a pesquisa e a decisão dos nodos a percorrer. Neste caso de estudo, a heurística utilizada foi a distância em linha reta desde o nodo atual até ao nodo destino, isto é, o depósito. Esta distância foi calculada tal como dito anteriormente através da Fórmula de Haversine, que tem como objetivo fornecer a distância entre dois pontos de uma esfera a partir das respetivas latitudes e longitudes.

Gulosa

O seguinte algoritmo que se segue é caracterizado por realizar a escolha que lhe parece melhor no momento, tendo em conta a estimativa calculada para cada nodo do grafo. Além disso, as escolhas que este toma são definitivas, não considerando as eventuais consequências de cada uma delas.

Considerando o limite máximo de capacidade do camião foi feito um predicado que apenas irá recomendar os circuitos cujo total de resíduos recolhidos seja abaixo dos 15 000L.

```
%-----------------------------------------------------------------------------------------------------------------------
% Pesquisa Informada - Algortimo da Pesquisa Gulosa
% resolve_gulosa(15885,C).

resolve_gulosa(Nodo,Caminho/Custo) :- estima(Nodo,Estima),
                                      agulosa([[Nodo]/0/Estima],InvCaminho/Custo/_),
                                      inverso(InvCaminho,Caminho).

agulosa(Caminhos, Caminho) :- obtem_melhor_g(Caminhos,Caminho),
                              Caminho = [Nodo|_]/_/_,goal(Nodo).

agulosa(Caminhos,SolucaoCaminho) :- obtem_melhor_g(Caminhos,MelhorCaminho),
                                    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
                                    expande_gulosa(MelhorCaminho,ExpCaminhos),
                                    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
                                    agulosa(NovoCaminhos,SolucaoCaminho).

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, ExpCaminhos) :- findall(NovoCaminho,adjacente2(Caminho,NovoCaminho),ExpCaminhos).
```

```
%-----------------------------------------------------------------------------
% Pesquisa Informada - Algortimo da Pesquisa Gulosa LIMITADA
% resolve_gulosaLimitada(15885,C).

resolve_gulosaLimitada(Nodo,Caminho/Custo) :- resolve_gulosa(Nodo,Caminho/Custo),
                                              ver_quant(Caminho,Quantidade),
                                              Quantidade =< 15000.
```

A* (A estrela)

O seguinte algoritmo trata-se de uma combinação de aproximações heurísticas como do algoritmo Breadth First Search (Busca em Largura) e da formalidade do Algoritmo de Dijkstra. Este apresenta um comportamento ótimo, quando é aplicada uma heurística apropriada.

Tal como foi feito no algoritmo da pesquisa Gulosa, foi também aplicado um predicado que irá dispensar todos aqueles circuitos que ultrapassam os 15 000L de resíduo urbano recolhido.

```
%-----------------------------------------------------------------------------------------------
% Pesquisa Informada - Algortimo da pesquisa A*
% resolve_aestrela(15885,C).

resolve_aestrela(Nodo,Caminho/Custo) :- estima(Nodo,Estima),
                                        aestrela([[Nodo]/0/Estima], InvCaminho/Custo/_),
                                        inverso(InvCaminho,Caminho).

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

aestrela(Caminhos,SolucaoCaminho) :- obtem_melhor(Caminhos,MelhorCaminho),
                                    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
                                    expande_aestrela(MelhorCaminho,ExpCaminhos),
                                    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
                                    aestrela(NovoCaminhos,SolucaoCaminho).

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) :- findall(NovoCaminho,adjacente2(Caminho,NovoCaminho), ExpCaminhos).
```

```
%-----------------------------------------------------------------------------------------------
% Pesquisa Informada - Algortimo da pesquisa A* LIMITADA
% resolve_aestrelaLimitada(15885,C).

resolve_aestrelaLimitada(Nodo,Caminho/Custo) :- resolve_aestrela(Nodo,Caminho/Custo),
                                                ver_quant(Caminho,Quantidade),
                                                Quantidade =< 15000.
```

Aplicação de Requerimentos

Identificar os circuitos com mais pontos de recolha por tipo de resíduo

Para calcular os circuitos com o maior número de pontos de recolha, foi criado um predicado capaz de devolver a lista mais cumprida dentro de uma lista que contém várias listas. De seguida é aplicado o predicado length, de forma, a ser conhecido o número de elementos que essa lista contém o que por sua vez corresponde ao número de pontos de recolha desse mesmo circuito representado pela lista. Como exemplo aplica-se o algoritmo DFS, mas é permitida a aplicação dos outros algoritmos desenvolvidos.

```
% ------------------------------------------------------------------------------------
% • Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher)
% maior_pontosR(15885,'all',_,Caminho,Len).
% maior_pontosR(15885,'seletivo','Lixos',Caminho,Len).
% maior_pontosR(15885,'seletivo','Papel e Cartão',Caminho,Len).
% maior_pontosR(15885,'seletivo','Vidro',Caminho,Len).

maior_pontosR(Nodo,Residuo,Tipo,L,Len) :- findall(Caminho, resolve_DFS(Nodo,Tipo,Residuo,Caminho,D,Q), Lista),
                                          maxList(Lista,L),
                                          length(L,Len).


maxList([X,Xs|T], Res) :-  length(X,Xr), length(Xs,Xsr),
                           Xr > Xsr,
                           maxList([X|T],Res).
maxList([X,Xs|T], Res) :-  length(X,Xr), length(Xs,Xsr),
                           Xr < Xsr,
                           maxList([Xs|T],Res).
maxList([X,Xs|T], Res) :-  length(X,Xr), length(Xs,Xsr),
                           Xr == Xsr,
                           maxList([Xs|T],Res).
maxList([L],L).
```

Comparar circuitos segundo o critério da distância e da quantidade recolhida

De forma a comparar circuitos tendo em conta os vários indicadores de produtividade existentes, foi desenvolvido o seguinte predicado, que poderá receber qualquer um dos algoritmos identificados por um número (0,1,2, tal como representado na figura). Este irá devolver a quantidade e distância de um circuito calculado por cada um dos algoritmos.

```
% ------------------------------------------------------------------------------------
% • Comparar circuitos de recolha tendo em conta os indicadores de produtividade ou seja a QUANTIDADE e a DISTANCIA
% 0- DFS
% 1- A*
% 2- Gulosa
% produtividade_circuito(15885,Dist,Quantidade,1).

produtividade_circuito(Nodo,Dist,Quantidade,0) :- resolve_DFS(Nodo,_,'all',[Nodo|Caminho],Dist,Quantidade).

produtividade_circuito(Nodo,Dist,Quantidade,1) :- resolve_aestrelaLimitada(Nodo,Caminho/Dist),
                                                  ver_quant(Caminho,Quantidade),!.

produtividade_circuito(Nodo,Dist,Quantidade,2) :- resolve_gulosaLimitada(Nodo,Caminho/Dist),
                                                  ver_quant(Caminho,Quantidade),!.

%...


ver_quant([],0).
ver_quant([X|Xs],Quantidade):- getQtLixo('all',_,QntLixo1,X),
                               ver_quant(Xs,QntLixo2),
                               Quantidade is QntLixo1 + QntLixo2.
```

Gerar circuitos de recolha tanto indiferenciada como seletiva que cubram um território

Para este requerimento foi aplicado o algoritmo DFS já desenvolvido anteriormente, adicionou-se apenas como parâmetro uma lista dos pontos de recolha nos quais o circuito deverá estar incluído. Portanto, confere-se se o nodo pelo qual o algoritmo pretende-se deslocar pertence ou não a essa mesma lista.

```
caminho_Territorio(Nodo,TipoLixo,Residuo,ListPontosT,[Nodo|Caminho]):-
                                        pp_Alg2(Nodo,[Nodo],Caminho,TipoLixo,Residuo,ListPontosT,15000).

pp_Alg2(Nodo,_,[],0,0,TipoLixo,Residuo,_,_):- goal(Nodo).

pp_Alg2(Nodo,Historico,[ProxNodo|Caminho],TipoLixo,Residuo,ListPontosT,Limite):-
                    Limite > 0,
                    adjacente(Nodo,ProxNodo,Dist1),
                    membro(ProxNodo,ListPontosT),
                    nao(membro(ProxNodo, Historico)),
                    getQtLixo(Residuo,TipoLixo,QntLixo1,Nodo),
                    Limite1 is Limite - QntLixo1,
                    pp_Alg2(ProxNodo,[ProxNodo|Historico],Caminho,QntLixo2,Dist2,TipoLixo,Residuo,ListPontosT,Limite1),
                    QntLixo is QntLixo1 + QntLixo2,
                    Dist is Dist1 + Dist2.
```

Cálculo do circuito mais rápido

Para o cálculo do circuito mais rápido considerou-se o circuito com a menor distância percorrida. Sendo assim utilizou-se o algoritmo já desenvolvido para os indicadores de produtividade, de forma a comparar os diferentes resultados para as várias pesquisas usadas.

```
% ------------------------------------------------------------------------------------------------------
% • Escolher o circuito mais rápido (usando o critério da distância);

circuito_rapidoPI1(Origem,Caminho,Dist) :- produtividade_circuito(Nodo,Dist,Quantidade,1).
circuito_rapidoPI2(Origem,Caminho,Dist) :- produtividade_circuito(Nodo,Dist,Quantidade,2).

circuito_rapidoPNI(Origem,Caminho,Dist) :- produtividade_circuito(Nodo,Dist,Quantidade,0).
```

Cálculo do circuito mais eficiente

No cálculo do circuito mais eficiente considerou-se o circuito que recolhe mais lixo por km percorrido. Daí o cálculo da quantidade a dividir pela distância. Mais uma vez usou-se o predicado desenvolvido para os indicadores de produtividade.

```
% ------------------------------------------------------------------------------------------------------
% • Escolher o circuito mais eficiente (usando um critério de eficiência à escolha);
% • Qual o circuito que recolhe mais lixo por km percorrido?
% circuito_eficientePI2(15885,Caminho,Taxa).

circuito_eficientePI1(Origem,Caminho,Taxa) :- produtividade_circuito(Nodo,Dist,Quantidade,1),
                                              Taxa is Quantidade / Dist.
circuito_eficientePI2(Origem,Caminho,Taxa) :- produtividade_circuito(Nodo,Dist,Quantidade,2),
                                              Taxa is Quantidade / Dist.

circuito_eficientePNI(Origem,Caminho,Taxa) :-  produtividade_circuito(Nodo,Dist,Quantidade,0),
                                               Taxa is Quantidade / Dist.
```

Análise de Resultados

Com a aplicação do predicado *statistics*, foi possível chegar aos seguintes resultados:

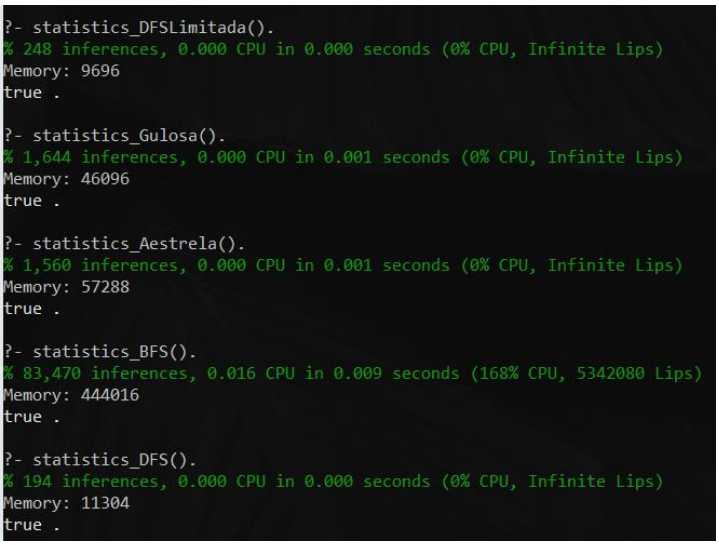
Estratégia\Critério	*Tempo (segundos)*	*Espaço*	*Profundidade/Custo*	*Encontrou melhor solução?*
DFS	0.000	11304	0.3076	Não
BFS	0.009	444016	40.3399	Não
Busca Iterativa Limitada em Profundidade	0.000	9696	0.3921	Não
Gulosa	0.001	46096	0.2698	Não
A (A estrela)*	0.001	57288	0.2277	Sim

Figura 1 - Tabela da análise final

Tal como é possível observar, o algoritmo A* é aquele que apresenta melhores resultados comparando o tempo, o espaço e o seu custo. Apresentando, portanto, um comportamento ótimo.

O algoritmo da pesquisa gulosa por sua vez, permitiu concluir que apesar da sua rápida execução, nem sempre conduz a soluções ótimas globais.

Já o algoritmo BFS, não apresenta bons resultados em termos de tempo e espaço.



Conclusão

Em suma, todas os requerimentos do trabalho prático foram desenvolvidos, tal como a comparação entre diversos algoritmos de pesquisa tanto informada como não informada.

Durante o projeto, uma das maiores dificuldades foi a interpretação do dataset dado, tendo em conta a necessidade de gestão de memória. Além disso, a adaptação dos algoritmos ao presente caso de estudo revelou-se também como um desafio.

Futuramente, poderiam ser explorados outros tipos de algoritmos ou então aprofundar os mesmos que foram elaborados neste trabalho prático, aplicando também outros requerimentos que se revelassem interessantes.