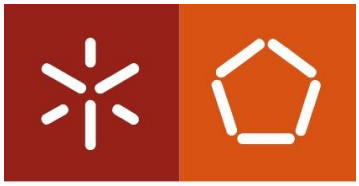

Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre - 2019/20)

Relatório SRCR - Trabalho Individual

A89501 Martim Bento da Rocha de Almeida

Braga

6 de junho de 2021

Resumo

Este relatório tem como objetivo descrever a solução criada para o problema proposto na cadeira de Sistemas de Representação de Conhecimento e Raciocínio.

O Trabalho consiste na representação e modelação do sistema de recolha de resíduos do concelho de Lisboa e na aplicação de algoritmos de pesquisa sobre a rede criada.

Foram utilizados algoritmos de pesquisa não informada e pesquisa informada, utilizando como heurísticas as informações apresentadas no dataset fornecido para o desenvolvimento do trabalho.

Uma vez que é utilizado esse dataset, é necessário o processamento do dataset de forma a ser possível representar num grafo a rede do circuito de recolha para, seguidamente, ser aplicado num conjunto de algoritmos implementados.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este trabalho teve como objetivo a representação do sistema de recolha de resíduos do concelho de Lisboa e na aplicação de algoritmos de pesquisa sobre a rede criada, recorrendo ao sistema de inferência do prolog como base da construção destes caminhos.

O sistema de recolha de resíduos consiste na existência de várias ruas ao longo do concelho que contém informação sobre o tipo de resíduo a ser recolhido tal como a sua quantidade. Esta informação encontra-se no dataset fornecido que, posteriormente, será processado e será extraída toda a informação pertinente para a construção da representação do sistema.

De forma a realizar a concepção dos percursos, foram utilizados vários algoritmos de pesquisa informada e de pesquisa não informada.

Capítulo 2

Descrição do Trabalho e Análise de Resultados

O trabalho está divido em seis diferentes componentes que representarão diferentes processos presentes no desenvolvimento do projeto.

O primeiro componente será o processo de **pré-processamento**, onde o dataset fornecido é processado e as decisões tomadas para a informação contida ser carregada para o prolog.

O componente do **carregamento dos dados** explica como a informação foi carregada para o prolog.

O componente da **formulação do problema** aborda superficialmente o problema apresentado.

O componente da **pesquisa não informada** implementará algoritmos de pesquisa não informada sobre o grafo resultante do processamento do dataset.

O componente da **pesquisa informada** implementará algoritmos de pesquisa informada sobre o grafo resultante do processamento do dataset, sendo que estes algoritmos utilizam heurísticas para obter os melhores caminhos.

O componente da **elaboração do caso prática** demonstra a utilização de algoritmos para obter determinadas informações.

2.1 Pré-processamento

Um ficheiro .xlsx como dataset foi fornecido para o desenvolvimento do projeto.

O ficheiro fornecido é referente a ruas e contêm as seguintes informações:

1. Latitude;
2. Longitude;
3. ObjectId;
4. Ponto de Recolha da Freguesia;
5. Ponto de Recolha Local;
6. O Tipo de resíduo encontrado no contentor;
7. O Tipo de contentor;
8. A capacidade do contentor;
9. A quantidade de contentores em cada ponto de recolha;
10. O Total de litros de contentores existente em cada ponto de recolha;

Para o desenvolvimento deste projeto, foi necessário que alguma informação fosse ser retirada ou modificada de forma a simplificar a representação do dataset num grafo. Esta informação foi processada principalmente de forma manual (teve o auxílio de Expressões Regulares, apesar de ter sido mínima, para filtragem de texto ao longo do dataset), não tendo a necessidade de utilização de nenhuma linguagem/biblioteca de manipulação de dados.

Assim, foi removida as seguintas informações:

- ObjectId, uma vez que será utilizada outro tipo de ID existente noutro campo de informação do dataset;
- Ponto de Recolha da Freguesia, uma vez que todas as ruas encontram-se na mesma freguesia é informação que se pode omitir;
- O Tipo de resíduo encontrado no contentor, considerando assim que todos os contentor têm o mesmo tipo de resíduos;
- O Tipo de Contentor, uma vez que é informação considerada irrelevante para o desenvolvimento deste projeto;
- Capacidade e Quantidade de Contentores em cada ponto de recolha, uma vez que já temos informação sobre a quantidade total em cada ponto de recolha.

O campo de informação sobre o Ponto de Recolha Local sofreu alterações. Estas alterações foram realizadas para simplificar a construção do grafo e para concatenar informações que estão contidas na mesma rua. O campo sobre o Ponto de Recolha Local é um campo que representa uma maior complexidade quando comparado com outros campos presentes no dataset. O campo contém as seguintes informações:

- Um ID específico;
- A Rua onde se encontra o ponto de recolha;
- Informação sobre a que ruas (pode não conter ruas ou conter no máximo duas ruas) se liga.

Com a especificação do campo realizada, as alterações realizadas foram:

- Criação de um campo ID;
- Criação de um campo Rua Ponto de Recolha;
- Criação de um campo Rua 1;
- Criação de um campo Rua 2;

2.1.1 ID

A criação deste campo tem como objetivo, concatenar sobre a informação da quantidade de litros de contentores num Ponto de Recolha. Uma vez que foi tomada a decisão de assumir que todo o resíduo tem o mesmo tipo, foi concatenada toda a informação sobre a quantidade de litros num Ponto de recolha, resultando assim na diminuição do números de linhas presentes no dataset. É de notar que com esta redução no número de linhas não implica a diminuição do conhecimento requerido para o desenvolvimento do projeto.

2.1.2 Rua Ponto de Recolha

Representa a rua onde se encontra toda a informação sobre a quantidade a ser recolhida.

2.1.3 Rua 1 e Rua 2

Representa ruas à qual a rua de ponto de recolha atual se liga. Estes dois campos terão utilidade aquando da criação da adjacência entre pontos de recolha, algo que será posteriormente explicado.

2.1.4 Adjacência

Para considerar um ponto de recolha adjacente a outro foram tomadas duas vertentes:

- Ligação entre ruas com o mesmo nome, sendo a ligação semelhante a uma estrutura de dados de Lista Ligada;
- Ligação entre ruas através dos campos Rua 1 e Rua 2.

Para ser explicado de forma simples a ligação através dos campos Rua 1 e Rua 2, será utilizado um caso existente no dataset processado:

Latitude	Longitude	ID	Rua Ponto Recolha Local	Rua 1	Rua 2	Quantidade Litros
-9.14348180670535	38.7073026157039	15807	R do Alecrim	R Ferragial	Pc Duque da Terceira	1700
-9.14361174262131	38.7067339965055	15890	Pc Duque da Terceira	R do Alecrim	R Remolares	1300

Como podemos verificar pela tabela, a linha da rua "R do Alecrim"tem conhecimento sobre a rua "Pc Duque da Terceira"no campo Rua 2 e a linha da rua "Pc Duque da Terceira"tem conhecimento sobre a rua "R do Alecrim"no campo Rua 1. Uma vez que têm conhecimento entre elas e essa informação encontra-se em campos Rua 1 e 2 contrários, é assumida a adjacência entre estes dois pontos de recolha.

Uma vez que este trabalho consta com dois pontos que não constam no dataset, sendo eles a garagem e o local de deposição dos resíduos, estes foram criados e possuem ligações a, pelo menos, uma rua.

```
rua(-9.13440,38.69784,0,"Garagem","Rua do Alecrim","",0).
move(0,15805,0.01357).
rua(-9.16239,38.71291,9999,"Depósito","R Remolares","Garagem",0).
move(15868,9999,0.01932).
move(9999,0,0.03179).
```

Figura 2.1: Criação dos locais Garagem e Local de Deposição

2.2 Carregamento dos dados

O carregamento dos dados da informação processada foi feita manualmente estando dividida em três ficheiros que são incluídos no ficheiro principal do projeto.

Os três ficheiros são:

- rua.pl, contém informação sobre as ruas do dataset;
- capacidade.pl, contém informação da capacidade em cada ponto de recolha;
- adjacencia.pl, contém informação sobre as ligações das ruas.

2.2.1 rua.pl

Toda a informação sobre a ruas foi guardada n predicado rua/7, tal como ficou resultante do processamento dos dados.

```
rua(-9.14330880914792,38.7080787857025,15805, "R do Alecrim", "R Ferragial", "R Ataíde",3390).
rua(-9.14337777820218,38.7080781891571,15806, "R do Alecrim" , "R Ataíde", "R Ferragial",3320).
rua(-9.14348180670535,38.7073026157039,15807, "R do Alecrim" , "R Ferragial", "Pc Duque da Terceira",1700).
rua(-9.14255098678099,38.7073286838222,15808, "R Corpo Santo" , "Lg Corpo Santo", "Tv Corpo Santo",2480).
rua(-9.14276596081499,38.7070836135523,15809, "Tv Corpo Santo" , "R Corpo Santo", "R Bernardino da Costa",400).
rua(-9.1426225690344,38.7066975168166,15810, "Tv Corpo Santo" , "R Bernardino da Costa", "Cais do Sodré",2160).
rua(-9.14240835587344,38.7069966274245,15811, "R Bernardino da Costa" , "Lg Corpo Santo", "Tv Corpo Santo",3000).
rua(-9.14305015543156,38.7068559589223,15812, "R Bernardino da Costa" , "Tv Corpo Santo", "Pc Duque da Terceira",3560).
rua(-9.1512511235953,38.7087754470349,15813, "Lg Conde-Barão" , "R da Boavista", "Bqr do Duro",6940).
rua(-9.15178946418214,38.7086356323308,15814, "Lg Conde-Barão" , "Bqr do Duro", "R Mastros",3020).
rua(-9.15201897924565,38.708606605818,15815, "Lg Conde-Barão" , "R Mastros", "Tv do Cais do Tojo",1280).
rua(-9.14910552718357,38.709073383915,15818, "Tv Marquês de Sampaio" , "R da Boavista", "R da Boavista",240).
rua(-9.14764976145157,38.708563591768,15819, "R da Boavista" , "R São Paulo", "Tv Marquês de Sampaio",3500).
rua(-9.1485717796828,38.7087267228466,15820, "R da Boavista" , "Tv Marquês de Sampaio", "Bqr dos Ferreiros",1410).
rua(-9.14911344583279,38.7088210959641,15821, "R da Boavista" , "Bqr dos Ferreiros", "Tv Marquês de Sampaio",850).
rua(-9.14949354638571,38.7088718347208,15822, "R da Boavista" , "Tv Marquês de Sampaio","Pto Galega (Rua da Boavista, 118)",1160).
rua(-9.15018388371526,38.7089108606806,15823, "R da Boavista" ,"Pto Galega (Rua da Boavista, 118)", "R Instituto Industrial",5190).
```

Figura 2.2: Predicado rua

2.2.2 capacidade.pl

Toda a informação sobre a capacidade encontra-se de uma forma simplificada no predicado capacidade/2 que contém a informação sobre a capacidade associada a um ID do Ponto de Recolha.

```
capacidade(15807,1700).
capacidade(15808,2480).
```

Figura 2.3: Predicado capacidade

2.2.3 adjacencia.pl

Toda a informação sobre as ligações entre pontos de recolha encontram-se no predicado move/3 que contém as ligações entre os IDs associados a pontos de recolha e a distância entre eles, sendo que essa distância é resultante do cálculo das distâncias euclidianas da Latitude e Longitude de dois pontos de recolha.

```
move(15820,15821,0.00055).
move(15821,15822,0.00038).
move(15822,15823,0.00069).
```

Figura 2.4: Predicado move

2.3 Formulação do Problema

O problema para resolver optado neste projeto foi a versão simplificada do problema, ou seja, os transportes responsáveis pela recolha têm capacidade ilimitada e, apesar de essa informação ter sido processada e carregada, a quantidade residual em cada ponto de recolha é desprezada.

Estado Inicial	Garagem
Estado Final	Garagem
Operadores	Predicado move/3
Custo	Distância Euclidiana entre as arestas

2.4 Pesquisa não-informada

Para a pesquisa não-informada, foi realizado o desenvolvimento de três algoritmos:

- Algoritmo Depth-First;
- Algoritmo Breadth-First;
- Algoritmo de Busca Iterativa em Profundidade (também conhecida como, Iterative Deepening Depth-First Search, ou IDDFS).

O algoritmo IDDFS revela ser bastante semelhante ao algoritmo Depth-First, contendo adicionalmente um limite do quão profundo no máximo se pode percorrer o grafo do projeto.

Posteriormente foi utilizado o Algoritmo Depth-First como base para aplicar aos requerimentos mínimos apresentados do trabalho.

Para o grafo resultante no projeto, o Algoritmo Breadth-First revelou ser o mais rápido sendo o Algoritmo de Busca Iterativa em Profundidade o mais lento. A utilização de memória entre os vários algoritmos mostrou não ter grandes diferenças. Estes resultados demonstraram ser supreendentes, uma vez que, reconhecendo estes algoritmos e o grafo deste projeto, não eram os resultados da qual esperava.

Estratégia	Complexidade Temporal	Complexidade Espacial	Encontrou a melhor solução?
DFS	$\theta(N^I)$	$\theta(NI)$	Sim
BFS	$\theta(N^I)$	$\theta(N^I)$	Sim
IDDFS	$\theta(N^I)$	$\theta(NI)$	Sim

2.4.1 Depth-First

```
depthFirst(Nodo, Destino, [Nodo|Caminho],Custo):-
    statistics(runtime,[Start|_]),
    depth(Nodo, Destino, [Nodo], Caminho2, Custo),
    length(Caminho2,C),
    C > 2,
    fim(F),
    completaCaminho(Caminho2,[F],Caminho),
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).

depth(Destino, Destino, _, [], 0).

depth(Nodo, Destino, Visited, [ProxNodo|Caminho],Custo):-
    adjacente(Nodo, ProxNodo,CustoRua),
    \+ member(ProxNodo, Visited),
    depth(ProxNodo, Destino, [Nodo|Visited], Caminho, CustoAcumulado),
    Custo is CustoRua + CustoAcumulado.
```

Figura 2.5: Algoritmo Depth-First

2.4.2 Breadth-First

```
breadthFirst(Nodo, Fim, Caminho, Custo):-
    statistics(runtime,[Start|_]),
    breadth(Fim, [[Nodo]], Caminho2),
    length(Caminho2,C),
    C > 2,
    calcularCusto(Caminho2,Custo),
    fim(F),
    completaCaminho(Caminho2,[F],Caminho),
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).

breadth(Fim, [[Fim|T]|_], Caminho):-
    inverso([Fim|T],Caminho).

breadth(Fim,[H|T],Caminho):-
    H = [Atual|_],
    findall([X|H],
        (Dest\==Atual,adjacente(Atual,X,_),\+ member(X,H)),
        Novos),
    append(T,Novos,Todos),
    breadth(Fim,Todos,Caminho).
```

Figura 2.6: Algoritmo Breadth-First

2.4.3 Iterative Deepening Depth-First Search

```
depthIterativeSearch(Nodo, Destino, [Nodo|Caminho],Custo, Altura):-
    statistics(runtime,[Start|_]),
    depthIterative(Nodo, Destino, [Nodo], Caminho2, Custo, Altura),
    length(Caminho2,C),
    C > 2,
    fim(F),
    completaCaminho(Caminho2,[F],Caminho),
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).

depthIterative(Destino,Destino,_,[],0,Altura):-
    Altura >= 0.

depthIterative(Nodo, Destino, Visited, [ProxNodo|Caminho], Custo, Altura):-
    adjacente(Nodo, ProxNodo,CustoRua),
    \+ member(ProxNodo, Visited),
    AlturaReduzida is Altura - 1,
    depthIterative(ProxNodo, Destino, [Nodo|Visited], Caminho, CustoAcumulado, AlturaReduzida),
    Custo is CustoRua + CustoAcumulado.
```

Figura 2.7: Algoritmo Iterative Deepening Depth-First Search

2.4.4 Predicados Auxiliares aos Algoritmos

```
adjacente(Nodo, NodoProx, Distancia):-
    move(Nodo, NodoProx, Distancia).

adjacente(Nodo, NodoProx, Distancia):-
    move(NodoProx, Nodo, Distancia).

completaCaminho(Caminho,Adicional,Final):-
    append(Caminho,Adicional,Final).

calcularCusto([_],0).
calcularCusto([],0).
calcularCusto([Nodo,NodoProx|T],Custo):-
    adjacente(Nodo,NodoProx,CustoLigacao),
    calcularCusto([NodoProx|T],CustoAcumulado),
    Custo is CustoLigacao + CustoAcumulado.
```

Figura 2.8: Predicado Auxiliares

2.5 Pesquisa informada

Para a pesquisa informada, foi realizado o desenvolvimento de dois algoritmos:

- Algoritmo Greedy;
- Algoritmo A*.

Este algoritmos requerem a utilização de heurísticas.

Neste projeto, a heurística utilizada foi a distância em linha reta entre dois pontos de recolha. Este valor é obtido através dos cálculos da distâncias euclidianas das latitudes e longitudes dos pontos de recolha.

O Algoritmo Greedy utiliza a heurística fornecida e escolhe o ponto com a heurística menor a cada iteração.

O Algoritmo A* utiliza a heurística fornecia e conta também a distância real para determinar qual ponto a escolher.

Em termos de resultados de execução, o Algoritmo A* revelou ser o algoritmo mais rápido a executar dos algoritmos de pesquisa informada e também revelou ser o mais eficiente em termos de utilização de memória.

Estratégia	**Complexidade Temporal**	**Complexidade Espacial**	**Encontrou a melhor solução?**
Greedy	$\theta(N^I)$	$\theta(N^I)$	Sim
A*	$\theta(N^I)$	$\theta(N^I)$	Sim

2.5.1 Algoritmo Greedy

```
resolveGulosa(Nodo, Destino, Caminho/Custo) :-
    statistics(runtime,[Start|_]),
    estima(Nodo, Destino, Estimativa),
    agulosa([[Nodo]/0/Estimativa], Destino, CaminhoInverso/Custo/_),
    inverso(CaminhoInverso, Caminho2),
    length(Caminho2,C),
    C >  2,
    fim(F),
    completaCaminho(Caminho2,[F],Caminho),
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).

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

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

expandeGulosa(Caminho, ExpCaminhos, Destino) :-
    findall(NovoCaminho, adjacenteG(Caminho, Destino,NovoCaminho), ExpCaminhos).
```

Figura 2.9: Algoritmo Greedy

2.5.2 Algoritmo A*

```
resolveAEstrela(Nodo, Fim, Caminho/Custo) :-
    statistics(runtime,[Start|_]),
    estima(Nodo, Destino, Estimativa),
    aestrela([[Nodo]/0/Estima], Fim, CaminhoInverso/Custo/_),
    inverso(CaminhoInverso, Caminho2),
    length(Caminho2,C),
    C > 2,
    fim(F),
    completaCaminho(Caminho2,[F],Caminho),
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).

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

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

expandeAEstrela(Caminho, Fim, ExpCaminhos) :-
findall(NovoCaminho, adjacenteG(Caminho,Fim,NovoCaminho), ExpCaminhos).

adjacenteG([Nodo|Caminho]/Custo/_, Fim, [ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
    adjacente(Nodo, ProxNodo, PassoCusto),\+ member(ProxNodo, Caminho),
    NovoCusto is Custo + PassoCusto,
```

Figura 2.10: Algoritmo A*

2.6 Elaboração do Caso Prático

Para além dos algoritmos implementados acima apresentados, através do algoritmo Depth-First foi realizado certas pesquisas para obter determinadas informações. As pesquisas são:

- Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território;
- Identificar qual o circuito com mais pontos de recolha;
- Escolher o circuito mais rápido.

Esteve também para ser realizado também a pesquisa pelo circuito mais eficiente. Uma vez que a formulação do problema é simplificada, o único critério de eficiência existente é a distância entre pontos de recolha, ou seja, a pesquisa pelo circuito mais eficiente está contido na pesquisa pelo circuito mais rápido.

2.6.1 Circuitos de Recolha Seletiva

A pesquisa pelo circuito de recolha seletiva é semelhante a uma pesquisa feita pelo algortimo Depth-First mas adiciona-se uma lista de pontos pela qual se deseja que o caminho resultante passe por. Para um caminho ser considerado válido, é necessário que passe por os pontos fornecidos na ordem pela qual são apresentados. Caso contrário, ficamos perante um caminho inválido.

```
depthSelectiveSearch(Nodo, Destino, [Nodo|Caminho], Custo, Pontos):-
    depthSelective(Nodo, Destino, [Nodo], Caminho2, Custo, Pontos),
    length(Caminho2,C),
    C > 2,
    fim(F),
    completaCaminho(Caminho2,[F],Caminho).

depthSelective(Destino, Destino, _, [], 0, Pontos).

depthSelective(Nodo, Destino, Visited, [ProxNodo|Caminho],Custo, Pontos):-
    adjacente(Nodo, ProxNodo,CustoRua),
    \+ member(ProxNodo, Visited),
    member(ProxNodo,Pontos),
    depthSelective(ProxNodo, Destino, [Nodo|Visited], Caminho, CustoAcumulado, Pontos),
    Custo is CustoRua + CustoAcumulado.
```

Figura 2.11: Predicado depthSelective

2.6.2 Circuito mais rápido

2.6.3 Circuito com mais pontos de recolha

Para descobrir o circuito com mais pontos de recolha bastou encontrar todas as soluções possíveis do grafo através do algoritmo Depth-First e filtra-se os resultados para obter o caminho em que a lista de pontos de recolha é o maior.

```
circuitoMaisPontos(Nodo, Destino, [Nodo|Caminho],Custo,L):-
    findall(([Nodo|Caminho],C),(depthFirst(Nodo, Destino, [Nodo|Caminho],Custo), length([Nodo|Caminho],C)),L2),
    maximo(L2,L).
```

Figura 2.12: Predicado circuitoMaisPontos

2.6.4 Circuito mais rápido

Para descobrir o circuito com mais rápido bastou encontrar todas as soluções possíveis do grafo através do algoritmo Depth-First e filtra-se os resultados para obter o caminho em que o custo de recolha total é o menor.

```
circuitoMaisRapido(Nodo, Destino, [Nodo|Caminho],Custo,L):-
    findall(([Nodo|Caminho],Custo),depthFirst(Nodo, Destino, [Nodo|Caminho],Custo),L2),
    minimo(L2,L).
```

Figura 2.13: Predicado circuitoMaisRapido

Capítulo 3

Conclusão

Em conclusão, foi um trabalho bem sucedido, tendo sido utilizado diferentes opções para a resolução de um problema específico, sendo que cada opções traz algumas adversidades como gestão de memória e tempo de execução.

Durante a execução deste projeto, as maiores dificuldades encontrada foram na construção e adaptação dos algoritmos ao grafo gerado e no processamento de dados, ou seja, entender quais os dados relevantes para o desenvolvimento deste projeto.

Futuramente, um trabalho adicional a fazer seria a implementação de mais algoritmos de pesquisa não-informada e informada, assim como a utilização de um dataset mais completo.