Universidade do Minho
Escola de Engenharia

Sistemas de Representação de Conhecimento e Raciocínio

Avaliação Individual

Autor
João Paulo Ribeiro Pereira

a89607

Mestrado Integrado em Engenharia Informática

3º Ano - 2º Semestre

Resumo

Este documento apresenta todo o processo de desenvolvimento de uma solução para o trabalho individual realizado no âmbito da UC de Sistemas de Representação de Conhecimento e Raciocínio. Serão detalhadas todas as estratégias utilizadas, desde a organização dos dados do problema até todas as abordagens feitas para resolver o problema e obter uma solução eficiente e válida.

O caso de estudo apresentado é a geração de circuitos de recolha de resíduos, em diferentes ruas da cidade de Lisboa. Através disso podem ser definidas várias funcionalidades tendo em conta indicadores de produtividade.

Depois da apresentação da estratégia de extração e organização dos dados provenientes dum dataset, é explicada a forma de como conhecimento armazenado e posteriormente todas as abordagens feitas para gerar os algoritmos mais indicados para cada funcionalidade.

No final apresentam-se os resultados e as conclusões.

Conteúdo

Lista de Figuras

1 Introdução

Este trabalho individual tem como objetivo apelar à formulação de problemas, desenvolvimento de técnicas e estratégias para resolver problemas, gerar algoritmos eficientes e utilizar técnicas de raciocínio perante as situações do problema real que enfrentamos.

Como base de trabalho possuímos um *dataset* fornecido pelos docentes, que contêm dados relativos aos circuitos e pontos de recolha de resíduos na cidade de Lisboa. O primeiro desafio é organizar os dados da melhor forma de maneira a gerar uma base de conhecimento que será utilizada no **Prolog** e dessa forma é possível trabalhar com esses dados e desempenhar as diferentes funcionalidades sobre os circuitos.

Em seguida o objetivo é utilizar os dados previamente organizados, nomeadamente, o grafo constituído pelos nodos e os arcos entre eles, com as informações necessárias para resolver os problemas que nos surgem.

Finalmente, depois de implementadas as funcionalidades e os algoritmos é feita uma reflexão e conclusão acerca dos resultados obtidos.

Foi-nos dada a escolha de optar por uma versão mais simplificada do problema, tendo apenas em conta as distâncias entre os pontos de recolha, e ignorando a capacidade do veículo coletor, assim como as quantidades recolhidas de cada tipo de resíduo, ou uma versão mais completa que inclui as quantidades e a tal capacidade. **Escolhi a versão simples** para elaborar este caso prático, no entanto **considerei as quantidades em situações específicas de algumas funcionalidades, que mencionarei mais abaixo.**

2 Apresentação e formulação do problema

O problema baseia-se em circuitos de recolha de resíduos, e desta forma temos sempre a seguinte constituição do circuito:

- **Percurso inicial** - Entre a garagem e o primeiro ponto de recolha.
- **Ponto de recolha** - Paragem num local para remover resíduos.
- **Entre pontos** - Entre locais de paragem.
- **Transporte** - Caminho entre a última paragem e o depósito.
- **Final** - Entre o depósito e a garagem.

Tendo em conta esta estrutura representativa do circuito, torna-se mais fácil formular o problema, porque já se conhece um pouco do comportamento que os algoritmos da solução vão ter.

Como formulação do problema, vou enunciar os dados iniciais, conjunto de estados, estados iniciais, finais, os operadores, as transições, o teste objetivo e por fim o custo da solução.

- **Dados iniciais** - Garagem e depósito, que serão o primeiro/último e penúltimo nodos a serem visitados no circuito, respetivamente. Outros dados são o tipo de resíduo em cada ponto de recolha, assim como as quantidades totais em litros desse resíduo, por fim, temos as distâncias entre o ponto de recolha atual e o próximo, armazenadas em cada nodo.
- **Conjunto de estados** - O problema representa o conjunto de estados com o identificador do ponto de recolha em que nos encontramos, de maneira geral. Por exemplo, a ligação entre a Rua Ferragial e Rua Ataíde foi interpretada como um mini-percurso de recolha, e possui o id 15805. Adicionalmente o estado também pode ser representado como um par (id,distância percorrida) ou (id,quantidade de resíduos recolhida), para as funcionalidades que utilizam essa informação, principalmente na pesquisa informada.
- **Estado inicial** - Garagem, dei a possibilidade de ser um nodo variável. Em maior parte dos testes começou com o id 15805, correspondente ao mini-percurso de recolha Rua Ferragial - Rua Ataíde.
- **Estado final** - Depósito, também defini como variável, através de um predicado no código, que pode tomar vários valores à escolha do utilizador, e em maior parte dos testes utilizei com o id 15899, que representa o mini-percurso que começa em Tv dos Remolares.

- **Operações** - Pré-condição - Acabar de chegar a um mini-percurso; Nome - Recolher resíduos; Efeito - O camião recolhe resíduos e avança para a próxima rua.
- **Transições** - Passar de um nodo para o próximo, com o objetivo de fazer uma nova recolha.
- **Teste objetivo** - Atingir o depósito ou visitar todos os nodos de um determinado circuito, utilizei algoritmos para estas duas diferentes abordagens.
- **Custo da solução** - Pode ser a distância exigida aquando da recolha de resíduos em cada mini-percurso, por exemplo o percurso com id 15805 exige um custo X, que é a distância entre o início desse percurso e o início de um dos próximos que lhe é adjacente.

  Em alguns casos também pode ser o custo de recolher X resíduos num dado mini-percurso, ou rua.

Depois de formular o problema desta forma foi mais fácil começar a organizar os dados e gerar a minha base de conhecimento. A primeira principal ideia foi a de representar cada nodo do grafo como um percurso de recolha afeto a uma determinada rua. Interpretei cada ligação entre ruas como um percurso de recolha, que contém informação sobre todos os contentores desse percurso, e a distância desse até ao próximo que lhe é adjacente.

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

Interpretei esta linha como sendo um nodo identificado por 15805, que começa em R Ferragial, e é adjacente a R Ataide que já diz respeito a um novo percurso com um novo identificador.

No *dataset* existem várias linhas com o mesmo identificador, mas juntei tudo num nodo e adicionei todos os contentores de resíduos que iam aparecendo, todos no mesmo nodo.

3 Extração e organização dos dados

Como referi anteriormente, antes de começar a transformar os dados estudei bem o *dataset* e encontrei uma solução para incorporar os dados necessários para criar o grafo numa base de conhecimento no **Prolog**.

Para realmente tratar dos dados criei um **script** em **Python**.

3.1 Estruturas utilizadas

Primeiro comecei por criar um dicionário (em que a chave é o id da rua) que armazena todos os nodos do meu grafo, neste caso são os mini-percursos, com início numa rua e fim noutra. Associado à chave fica a rua início, rua fim, e a localização relativa ao início desse percurso, entre o inicio e fim:

```
ruasDict.update({idRuas : (inicio, fim, localizacao)})
```

Depois, noutro dicionário, associei ao id da rua, os contentores que aparecem a ele associados durante a leitura do *dataset*, com o par (lixo,litros) em que lixo designa o tipo de resíduo (Lixos, Papel e Cartao, etc) e litros designa o total em litros, que corresponde à última coluna do *xlsx*:

```
par = (lixo,litros)
...
residuosDict[idRuas].append(par)
```

O próximo passo foi arranjar maneira de criar os arcos e de os armazenar também num último dicionário, de acordo com os nodos que já tinha armazenados, decidi fazer da seguinte forma: numa combinação Rua Inicio - Rua fim de um dado percurso, procurei todas as ocorrências da Rua fim, como sendo a rua início de outro percurso, e formei esses arcos. Ou seja, para o exemplo (15805) Rua Ferragial - Rua Ataíde, supondo que temos a ligação (15806) Rua Ataíde - Rua Ferragial, e também (15807) Rua Ferragial - Pc Duque da Terceira, liga-se primeiramente o nodo 15805 ao 15806, e depois fazemos o mesmo processo, avançando para o 15806, e ligamos este ao 15805 e ao 15807, porque a rua destino deste é rua início dos outros. Cada arco é constituido pelo id do percurso inicio, id do percurso destino e a distância entre eles, apresentada em metros.

```
arcosDict[idInicio].append((idInicio, idDest, dist))
```

Com tudo isto feito, depois gerei dois ficheiros **.pl** para dar import na minha **main.pl**, que é o meu programa principal do **Prolog**.

Esses ficheiros são **nodos.pl** e **arcos.pl**, que me armazenam a informação sobre um nodo (idRua, NomeInicio, (Latitude, Longitude),[(Lixo,TotalLitros)]) e sobre os arcos (idInicio, idDestino, Distancia), respetivamente. Com isto feito fiquei com 78 nodos e 180 arcos. A garagem por omissão, é gerada através do parser como situada no percurso com id 15805, e o depósito no percurso com id 15899, mas depois existe um predicado (inicial(X) e final(Y)) no programa principal que permite alterar estes locais.

Também gerei um arco fixo, que nos dá caminho direto do depósito para a garagem, já que no caminho de volta não há restrições, pois optei pela versão simples sem capacidade máxima de recolha, e por isso a recolha é feita toda duma vez. A distância deste arco é a distância em linha reta entre o depósito e a garagem, por considerar que existe um caminho direto.

Exemplos dos ficheiros **.pl**:

- **arcos.pl**:

```
arco(15878,15812,54.0582503922).
arco(15851,15849,69.8391628137).
arco(15887,15880,206.171730106).
arco(15820,15848,134.006943938).
arco(15846,15847,111.238855901).
arco(15849,15834,370.533413579).
arco(15834,15849,370.533413579).
```

- **nodos.pl**:

```
nodo(15842,'R Ribeira Nova',(-9.14614421827306, 38.707783978131),[('Lixos', 90), ('Lixos', 90)]).
nodo(15834,'Av 24 de Julho',(-9.15046432876378, 38.7068005850863),[('Lixos', 140), ('Lixos', 1920)]).
```

3.2 Configuração do ambiente de trabalho

Já tendo a base de conhecimento formada com os ficheiros **nodos.pl** e **arcos.pl** para representar o grafo, criei também um **auxiliares.pl** para todos os predicados mais simples, que servem como auxílio aos algoritmos desenvolvidos.

Alguns deles foram baseados nos das aulas práticas mas adaptados a este caso de estudo.

No ficheiro principal **main.pl** ficam todos os algoritmos fundamentais para representar a solução ao problema da geração de circuitos de recolha de resíduos, e para começar, defini os dados essenciais:

```
% Estado inicial (Garagem)
inicial(15805).

% Estado final (Deposito)
final(15899).

% LIMITE PARA A PESQUISA EM PROFUNDIDADE
limite(6).

% Numero de pontos de recolha do grafo
numNodos(6).

% Tipo de resduo a recolher
tipo('Lixos').
```

A garagem e o depósito ficam representados por inicial e final, e depois tenho o predicado limite, para limitar a pesquisa em profundidade, depois o numNodos, que limita a quantidade de nodos que exigo ao algoritmo visitar, é utilizado para efeitos de teste, e indica que o algoritmo só deve acabar depois de passar em 6 percursos de recolha durante o circuito. Por fim, o predicado tipo é utilizado para gerar circuitos seletivos, para recolher só resíduos de um certo tipo.

4 Funcionalidades

Antes de explicar as estratégias de procura utilizadas, é importante enumerar e salientar as funcionalidades implementadas no programa, assim como todos os tipos de pesquisas, testes e visualização de circuitos a que o utilizador pode ter acesso.

O programa contém predicados que permitem representar conhecimento e raciocinar sobre o grafo gerado de modo a cumprir estes requisitos:

1. Gerar circuitos de recolha num dado território, tendo em conta o local da garagem e do depósito (nodos inicial e final), limitado pelo número de percursos de recolha que queremos visitar (nodos), ou seja, se quisermos visitar 10 pontos de recolha, é possível gerar circuitos que visitem 10 nodos, e chegando ao $10^{\underline{o}}$, sigam o primeiro caminho encontrado até ao depósito, e depois até à garagem. Nesta funcionalidade é dada informação da distância total percorrida e a quantidade de litros de lixo recolhida (tendo em conta que a capacidade do camião é ilimitada).

2. Obter uma lista de todos os circuitos de recolha possíveis, que visitem X nodos, dada uma garagem, e um depósito.

3. Gerar qualquer circuito de recolha, desde que atinja o depósito.

4. Obter uma lista dos circuitos enunciados imediatamente em cima.

5. Gerar qualquer circuito de recolha, mas seletivo, ou seja, que só recolha um determinado tipo de resíduo, e deixe os outros no contentor, e indicando a distância percorrida e quanto se recolheu em cada ponto.

6. Obter uma lista de todos os circuitos para o que foi enunciado imediatamente acima.

7. Gerar quaisquer circuitos, tanto seletivos como indiferenciados, sem exigências de visitar X nodos, mas com limitação de pesquisa em profundidade (predicado limite), indicando a distância percorrida.

8. Encontrar o circuito que contém mais pontos de recolha de um certo tipo (mais contentores desse tipo ao longo do circuito), indicando o número de pontos de recolha que se atingiu.

9. Gerar vários circuitos tendo em conta os indicadores de produtividade, só em função da distância, ou então tendo em conta a distância e a quantidade recolhida, utilizando uma heurística *distancia/quantidade*. Apresenta a quantidade total recolhida e a distância percorrida.

10. Escolher o circuito mais rápido, em função do que percorre menos metros. Apresenta a distância percorrida.

11. Escolher o circuito mais eficiente, encontrando a razão *distancia/quantidade* mais baixa possível. Apresenta a razão mais baixa encontrada e o circuito gerado.

5 Estratégias de procura

Para implementar todas as funcionalidades acima descritas, utilizei pelo menos uma vez cada tipo de pesquisa sugerida pelos docentes, no entanto não utilizei todos os tipos para todas as funcionalidades. Em certas funcionalidades achei mais intuitivo, mais prático ou mais indicado utilizar uma estratégia e não as outras.

5.1 Procura não informada

Este tipo de procura funcionou bem para casos em que o número de circuitos gerados era muito grande, e como o grafo gerado a partir do *parser* ainda era um pouco grande, tornou-se uma vantagem usar mais vezes a procura não informada, principalmente a *Depth First Search*. No entanto um dos problemas é que nos encontra o primeiro caminho em profundidade, inicialmente, e para funcionalidades que exigem o ótimo, não foi o mais indicado, porque também não tira partido das informações de estado do grafo, é tudo *às cegas*.

5.1.1 *Depth-First Search*

Esta foi a estratégia que mais utilizei para gerar todos os circuitos possíveis, por exemplo, porque lida bem com muitas soluções. Foi utilizada para gerar um caminho qualquer desde um ponto até ao depósito, e por isso o caso de paragem utilizado foi atingir o depósito:

```
?- dfs(15808,S,Dist).
S = [15808/'Lg Corpo Santo',
15879/'Tv Corpo Santo',
15811/'Lg Corpo Santo',
15812/'Tv Corpo Santo',
15898/'Pc Duque da Terceira',
15866/'Tv dos Remolares',
15890/'R do Alecrim', 15891/'R Remolares', ... / ...|...],
Dist = 5094.839905557002.
```

Também se mostrou útil para gerar um circuito de recolha que visitasse pelo menos X nodos indicados no predicado limite (neste caso são 6 e depois de visitar 6, ele gera automaticamente o caminho de volta à garagem, passando no depósito 15899), aqui o caso de paragem é verificar se pelo menos 6 nodos já estão no histórico do algoritmo, neste caso a numeração só vai até 5, porque o nodo 6 já entra no caminho gerado para ir até ao depósito:

```
?- printDfsCircuitoRecolha(15805).
1) Id: 15805 | Rua: R Ferragial | Total recolhido: 3390 litros
2) Id: 15806 | Rua: R Ataide | Total recolhido: 3220 litros
3) Id: 15807 | Rua: R Ferragial | Total recolhido: 1700 litros
4) Id: 15898 | Rua: Pc Duque da Terceira | Total recolhido: 470 litros
```

```
5) Id: 15866 | Rua: Tv dos Remolares | Total recolhido: 6040 litros
1) Id: 15867 | Rua: R do Alecrim
2) Id: 15812 | Rua: Tv Corpo Santo
3) Id: 15898 | Rua: Pc Duque da Terceira
4) Id: 15866 | Rua: Tv dos Remolares
5) Id: 15890 | Rua: R do Alecrim
6) Id: 15891 | Rua: R Remolares
7) Id: 15884 | Rua: Av 24 de Julho
8) Id: 15851 | Rua: R Ribeira Nova
9) Id: 15849 | Rua: R Dom Luis I
10) Id: 15876 | Rua: Av 24 de Julho
11) Id: 15877 | Rua: R Cintura (Santos)
12) Id: 15868 | Rua: Pc Duque da Terceira
13) Id: 15869 | Rua: Tv dos Remolares
14) Id: 15887 | Rua: Tv Ribeira Nova
15) Id: 15880 | Rua: R Sao Paulo
16) Id: 15885 | Rua: R Sao Paulo
17) Id: 15843 | Rua: Tv Carvalho
18) Id: 15857 | Rua: Pc Sao Paulo
19) Id: 15899 | Rua: Tv dos Remolares
20) Id: 15805 | Rua: R Ferragial
Total recolhido no circuito: 14820 |
Distancia total percorrida: 5163.868362585962 metros
```

A versão de encontrar todos os caminhos com esta limitação de visitar X nodos não deu para testar, porque o grafo tem muitas possibilidades de soluções, principalmente na parte de voltar para a garagem.

Também foi utilizada esta estratégia para gerar todos os circuitos que partem da garagem, e o resultado foram 2042 circuitos diferentes, desde aquele ponto:

```
?- qualquerCaminhoIndiferenciado(S,N).
S = [([15805/'R Ferragial'/3390, 15806/'R Ataide'/3220,
15807/'R Ferragial'/1700, 15898/'Pc Duque da Terceira'/470,
15866/'Tv dos Remolares'/6040, ... / ... / 2640, ... / ...|...], 16),
([15805/'R Ferragial'/3390, 15806/'R Ataide'/3220,
15807/'R Ferragial'/1700, 15898/'Pc Duque da Terceira'/470, ... / ... / 6040, ..
([15805/'R Ferragial'/3390,
15806/'R Ataide'/3220,
15807/'R Ferragial'/1700, ... / ... / 470, ... / ...|...], 17),
([15805/'R Ferragial'/3390,
15806/'R Ataide'/3220, ... / ... / 1700, ... / ...|...], 17),
([15805/'R Ferragial'/3390, ... / ... / 3220, ... / ...|...], 15),
([... / ... / 3390, ... / ...|...], 15), ([... / ...|...], 16),
([...|...], 16), (..., ...)|...],
N = 2042.
```

Agora sem limitação de visitar X nodos, podemos utilizar este algoritmo para gerar um caminho seletivo, de acordo com o tipo pedido, neste exemplo foi *Papel e Cartao*, que é dos menos frequentes de todos os pontos de recolha:

```
?- printDfsSeletiva(15805).
1) Id: 15805 | Rua: R Ferragial | Total recolhido: 1530 litros
2) Id: 15806 | Rua: R Ataide | Total recolhido: 460 litros
3) Id: 15807 | Rua: R Ferragial | Total recolhido: 380 litros
4) Id: 15898 | Rua: Pc Duque da Terceira | Total recolhido: 0 litros
5) Id: 15866 | Rua: Tv dos Remolares | Total recolhido: 480 litros
6) Id: 15867 | Rua: R do Alecrim | Total recolhido: 0 litros
7) Id: 15812 | Rua: Tv Corpo Santo | Total recolhido: 0 litros
8) Id: 15868 | Rua: Pc Duque da Terceira | Total recolhido: 280 litros
9) Id: 15869 | Rua: Tv dos Remolares | Total recolhido: 0 litros
10) Id: 15887 | Rua: Tv Ribeira Nova | Total recolhido: 0 litros
11) Id: 15880 | Rua: R Sao Paulo | Total recolhido: 0 litros
12) Id: 15885 | Rua: R Sao Paulo | Total recolhido: 0 litros
13) Id: 15843 | Rua: Tv Carvalho | Total recolhido: 370 litros
14) Id: 15857 | Rua: Pc Sao Paulo | Total recolhido: 140 litros
15) Id: 15899 | Rua: Tv dos Remolares | Total recolhido: 0 litros
16) Id: 15805 | Rua: R Ferragial | Total recolhido: 0 litros
Total recolhido no circuito: 3640 |
Distancia percorrida: 1398.86035602866 metros
true
```

Para encontrar o circuito com mais pontos de recolha de um certo tipo também utilizei este algoritmo, e deu o seguinte *output*:

```
?- maisPontosRecolha.
1) Id: 15805 | Rua: R Ferragial | Num Pontos: 2
2) Id: 15806 | Rua: R Ataide | Num Pontos: 3
3) Id: 15807 | Rua: R Ferragial | Num Pontos: 2
4) Id: 15898 | Rua: Pc Duque da Terceira | Num Pontos: 0
5) Id: 15866 | Rua: Tv dos Remolares | Num Pontos: 1
6) Id: 15890 | Rua: R do Alecrim | Num Pontos: 0
7) Id: 15891 | Rua: R Remolares | Num Pontos: 1
8) Id: 15884 | Rua: Av 24 de Julho | Num Pontos: 1
9) Id: 15851 | Rua: R Ribeira Nova | Num Pontos: 1
10) Id: 15849 | Rua: R Dom Luis I | Num Pontos: 0
11) Id: 15876 | Rua: Av 24 de Julho | Num Pontos: 0
12) Id: 15877 | Rua: R Cintura (Santos) | Num Pontos: 0
13) Id: 15878 | Rua: Pc Duque da Terceira | Num Pontos: 0
14) Id: 15879 | Rua: Tv Corpo Santo | Num Pontos: 1
15) Id: 15811 | Rua: Lg Corpo Santo | Num Pontos: 1
16) Id: 15812 | Rua: Tv Corpo Santo | Num Pontos: 0
17) Id: 15868 | Rua: Pc Duque da Terceira | Num Pontos: 1
18) Id: 15869 | Rua: Tv dos Remolares | Num Pontos: 0
19) Id: 15887 | Rua: Tv Ribeira Nova | Num Pontos: 0
20) Id: 15880 | Rua: R Sao Paulo | Num Pontos: 0
21) Id: 15885 | Rua: R Sao Paulo | Num Pontos: 0
22) Id: 15843 | Rua: Tv Carvalho | Num Pontos: 2
```

```
23) Id: 15859 | Rua: Pc Sao Paulo | Num Pontos: 1
24) Id: 15865 | Rua: Pc Sao Paulo | Num Pontos: 2
25) Id: 15805 | Rua: R Ferragial | Num Pontos: 0
26) Id: 15899 | Rua: Tv dos Remolares | Num Pontos: 0
Total de pontos do tipo Papel e Cartao 19
true
```

Para encontrar o caminho mais rápido, com a menor distância, também foi útil este algoritmo, utilizando também o *findall* e a função mínimo:

```
?- maisRapido.
1) Id: 15805 | Rua: R Ferragial | Distancia da rua: 7.67169611396
2) Id: 15806 | Rua: R Ataide | Distancia da rua: 85.9531812876
3) Id: 15807 | Rua: R Ferragial | Distancia da rua: 143.058059511
4) Id: 15898 | Rua: Pc Duque da Terceira | Distancia da rua: 40.4629809401
5) Id: 15899 | Rua: Tv dos Remolares | Distancia da rua: 0
6) Id: 15805 | Rua: R Ferragial |
Distancia da rua: 0
Distancia otima: 277.14591785266003
true
```

Por fim, também experimentei utilizar para o caminho mais eficiente, em termos da razão *distancia/quantidadeRecolhida*:

```
?- maisEficiente.
1) Id: 15805 | Rua: R Ferragial | Razao de eficiencia: 0.0022630372017581122
2) Id: 15806 | Rua: R Ataide | Razao de eficiencia: 0.02669353456136646
3) Id: 15807 | Rua: R Ferragial | Razao de eficiencia: 0.08415179971235295
4) Id: 15898 | Rua: Pc Duque da Terceira | Razao de eficiencia: 0.08609144880872
5) Id: 15899 | Rua: Tv dos Remolares | Razao de eficiencia: 0
6) Id: 15805 | Rua: R Ferragial |
Razao de eficiencia: 0
Razao otima: 0.19919982028420094
true
```

Como se pôde ver, utilizei este algoritmo em quase todas as funcionalidades por ser dos que lida melhor com muitas soluções, no entanto para conseguir testar, por vezes tive de descartar alguns caminhos, e por isso a eficiência e as soluções ótimas não são garantidas como as melhores.

5.1.2 *Depth-Limited Search*

Esta estratégia decidi utilizar pouco, não achei tão fundamental para as funcionalidades no âmbito geral, no entanto quando utilizei foi para comparar e testar em relação à procura em profundidade original. Foi implementada para gerar os circuitos indiferenciados, ou seletivos, sem restrição de visitar pelo menos X nodos. O limite utilizado aqui foi de 10 nodos e a seleção por papel e cartão. É de notar que as distâncias de ida ao depósito e volta à garagem não estão a ser contadas para comparar estas funcionalidades, mas esse percurso podia depois ser gerado com o algoritmo *Depth-first search* para o efeito.

```
?- printDfsLimitada(15805).
1) Id: 15805 | Rua: R Ferragial | Total recolhido: 3390 litros
2) Id: 15806 | Rua: R Ataide | Total recolhido: 3220 litros
3) Id: 15807 | Rua: R Ferragial | Total recolhido: 1700 litros
4) Id: 15898 | Rua: Pc Duque da Terceira | Total recolhido: 470 litros
5) Id: 15866 | Rua: Tv dos Remolares | Total recolhido: 6040 litros
6) Id: 15867 | Rua: R do Alecrim | Total recolhido: 2640 litros
7) Id: 15812 | Rua: Tv Corpo Santo | Total recolhido: 3560 litros
8) Id: 15868 | Rua: Pc Duque da Terceira | Total recolhido: 4260 litros
9) Id: 15869 | Rua: Tv dos Remolares | Total recolhido: 1600 litros
10) Id: 15887 | Rua: Tv Ribeira Nova | Total recolhido: 480 litros
Total recolhido no circuito: 27360 |
Distancia percorrida: 890.8830766155601 metros
true

?- printDfsSeletivaLim(15805).
1) Id: 15805 | Rua: R Ferragial | Total recolhido: 1530 litros
2) Id: 15806 | Rua: R Ataide | Total recolhido: 460 litros
3) Id: 15807 | Rua: R Ferragial | Total recolhido: 380 litros
4) Id: 15898 | Rua: Pc Duque da Terceira | Total recolhido: 0 litros
5) Id: 15866 | Rua: Tv dos Remolares | Total recolhido: 480 litros
6) Id: 15867 | Rua: R do Alecrim | Total recolhido: 0 litros
7) Id: 15812 | Rua: Tv Corpo Santo | Total recolhido: 0 litros
8) Id: 15868 | Rua: Pc Duque da Terceira | Total recolhido: 280 litros
9) Id: 15869 | Rua: Tv dos Remolares | Total recolhido: 0 litros
10) Id: 15887 | Rua: Tv Ribeira Nova | Total recolhido: 0 litros
Total recolhido no circuito: 3130 |
Distancia percorrida: 890.8830766155601 metros
true
```

5.1.3 *Breadth-First Search*

Apenas utilizei esta estratégia para tentar encontrar a solução ótima em termos de distância, ou seja, encontrar a solução mais rápida e com a menor distância. Neste método todos os sucessores de um nodo são explorados primeiro e a solução é ótima se o custo da solução for 1, neste caso o custo pode ser comparado a 1 porque a pesquisa é não informada e o camião apenas avança de nodo em nodo.

A solução obtida foi a seguinte:

```
?- bfsIndiferenciada(S).S = [(15806, 'R Ataide', 7.67169611396),
(15807, 'R Ferragial', 85.9531812876), (15868, 'Pc Duque da Terceira',
75.6091106509), (15899, 'Tv dos Remolares', 93.1954569331)].
```

Neste resultado o circuito vai da garagem para o percurso 15806, e acaba no depósito muito rapidamente, é uma boa candidata a solução ótima, porque foi a mais baixa que consegui encontrar, e neste caso a distância do nodo, é o total até chegar a ele, diferindo assim dos outros *outputs*.

5.2 Procura informada

A pesquisa informada foi melhor para as funcionalidades de eficiência e dos melhores caminhos, porque neste caso podemos contar com informações do grafo, como distância a que estamos do próximo nodo, a quantidade de pontos de recolha que há nos próximos nodos, entre outros. O critério que utilizei para a gulosa, foi tentar tirar partido das distâncias dos nodos vizinhos ou do número de pontos de recolha, para atingir o ótimo, enquanto que na A estrela, utilizei os dois critérios em conjunto através da razão *distancia/quantidade*.

5.2.1 *Greedy Search*

Pesquisa gulosa para procurar imediatamente o nodo mais próximo que satisfaça o requisito que queremos, por exemplo, o nodo adjacente que contém mais pontos de recolha:

```
?- resolve_gulosa(15805).
15805
15806
15807
15868
15899
Numero de pontos de recolha: 10
true
```

Pode reparar-se que o primeiro caminho que nos dá só tem 10 pontos de recolha, e se fizermos *backtracking* há mais, isto porque no primeiro passo podemos ser gulosos, mas nos próximos acabar por perder, e não atingir o ótimo, por isso esta aplicação da gulosa não serve para este problema.

Outra aplicação da gulosa ocorre em descobrir o caminho mais rápido, no primeiro resultado obtém-se:

```
?- maisRapidoGuloso.
15805
15806
15807
15868
15899
Distancia total percorrida: 270.10114109952
true
```

5.2.2 A*

Esta estratégia só apliquei para o caminho mais eficiente para tirar partido da distância mínima e quantidade recolhida num ponto, em vez de utilizar estimativa da distância que falta, utilizei a minimização da razão *distancia/quantidade*:

```
?- resolve_aestrela(15805).
15805
15806
15807
15868
15866
15867
15812
15898
15899
Distancia percorrida: 521.55737460836 |
Quantidade recolhida: 25280
true
```

6 Resultados

Depois de analisar todos os *outputs* obtidos em cima, poderá dizer-se que as melhores soluções envolvem o caminho 15805 - 15806 - 15807 - 15868 - 15899 - 15805, com uma distância de 270 metros aproximada e o caminho 15805 - 15806 - 15898 - 15899 - 15805.

Como o grafo é muito grande, as soluções ótimas podem não ter sido captadas pelo algoritmo dfs, pois para o utilizar foram reduzidos vários caminhos da lista dos possíveis, no entanto com a gulosa e bfs parece que atingimos a melhor solução de todas, em termos de distância.

Estratégia	Tempo(s)	Espaço	Profundidade/Custo	Melhor Solução?
DFS	20.956	O(bm)	277	Não
BFS	0.01	$O(b^{d+1})$	270	Sim
DFS-LIM	-	O(bl)	-	Não
GREEDY	0.001	$O(b^{m})$	270	Sim
A*	26.812	Nodos com g(n)+h(n)	521	Não

- b - fator de ramificação
- m - máxima profundidade da árvore
- l - limite de pesquisa
- d - profundidade da solução

Esta análise de resultados foi feita comparando apenas a distância como indicador de produtividade, a heurística da abordagem A estrela não foi a mais indicada para esta situação, visto que não consegui atingir a solução ótima, nem estimar as distâncias até ao final do percurso. Em contrapartida, a heurística da gulosa pareceu-me a ideal para obter a solução ótima, é de notar que em algumas estratégias não consegui testar todos os caminhos possíveis devido à complexidade do grafo.

Relativamente à procura em profundidade limitada, também não incluí nesta funcionalidade do caminho mais rápido, porque não consegui implementar as tentativas por limite para encontrar o caminho, a única coisa que utilizei foi um limite fixo para testes na funcionalidade de gerar qualquer caminho que atinja o depósito.

7 Comentários finais e conclusão

Este projeto individual tornou-se um grande desafio, tanto na descoberta da melhor forma de gerar o grafo, como também na manipulação e desenvolvimento das soluções através dos algoritmos de pesquisa.

Surgiram muitos problemas de demoras, ou erros da *stack* ao correr os algoritmos, e deste modo necessecitei de adaptar o contexto do meu grafo de forma a conseguir testar, e infelizmente tive de excluir alguns arcos para conseguir obter resultados.

Além disso, a heurística da A estrela, que parecia ser a mais indicada neste contexto, revelou-se fraca, pois não me deu a solução ótima. A procura em profundidade revelou-se poderosa neste grafo, mas mesmo assim não chegou à solução ótima porque talvez um dos caminhos descartados podia ser essa solução. A bfs deu-me o caminho ótimo relativamente à minimização da distância, o que é positivo. A gulosa revelou-se também muito útil e foi bem conseguida. Os grandes pontos a melhorar seriam a heurística A estrela, e a organização dos dados de modo a obter soluções mais completas do grafo, sem ter de descartar caminhos.

No entanto, acho que consegui apresentar um programa útil para verificar maior parte dos caminhos existentes, tendo em conta o tipo de resíduo a recolher, as distâncias e quantidades possíveis de recolha, foquei-me em aproximar o grafo do real, com mais arcos, mas também sem exagerar com medo de não conseguir correr nenhum dos algoritmos, acho que este balanço foi bem conseguido porque no final apresentei alguns resultados capazes de revelar as candidatas às soluções ótimas, nomeadamente no contexto da distância e caminho mais rápido.