Universidade do Minho
Escola de Engenharia

UNIVERSIDADE DO MINHO

TRABALHO INDIVIDUAL

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

(2º SEMESTRE 20/21)

A89506 - Luís Miguel Lopes Pinto

Braga

Junho - 2021

RESUMO

O seguinte relatório visa abordar algumas das temáticas lecionadas no âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio, em particular, serviu para aprofundar os conhecimentos sobre métodos de resolução de problemas e de procura.

O caso de estudo baseou-se nos circuitos de recolha de resíduos urbanos do concelho de Lisboa. De forma a atingir os objetivos propostos, foi fundamental o uso de mecânicas auxiliares para extração de dados do dataset fornecido, a implementação de algoritmos de pesquisa não informada e informada sobre o grafo criado e a utilização de heurísticas tendo em conta coordenadas geográficas dos contentores de lixo, da garagem e do depósito.

Foi proposta a utilização da linguagem de programação em logica PROLOG que permitiu realizar todas as tarefas propostas. Para além disso, em concordância com o supra mencionado, a da base de conhecimento utilizada derivou do uso de mecânicas auxiliares sobre o dataset fornecido e para isso foi utlizada linguagem de programação JAVA.

De modo a demonstrar os resultados obtidos e o funcionamento do trabalho realizado foi elaborada a secção análise de resultados. Por último na secção de conclusões e sugestões é feita uma análise crítica do trabalho realizado bem como sugestões a adotar em futuras evoluções do trabalho.

ÍNDICE

ÍNDICE FIGURAS

ÍNDICE TABELAS

INTRODUÇÃO

O projeto individual tem como principal objetivo construir um sistema de recolhas de resíduos urbanos. Em adição, permitiu estimular o uso de técnicas de formulação de problemas, a aplicação de diversas estratégias para a resolução de problemas com o uso de algoritmos de procura e o desenvolvimento de mecanismos de raciocínio adequados a esta problemática. Na solução implementada optou-se pela versão completa.

De acordo com a proposta de criação de circuitos de recolha de resíduos urbanos, foi fornecido um dataset contendo informações relativas à periodicidade e horário de início da recolha, detalhe de cada ponto abrangido pelo circuito no que diz respeito a localização, capacidade e tipo de contentor lá existente, entre outras.

Para além do dataset, foram ainda fornecidas algumas considerações, nomeadamente: a noção de circuito, a existência de uma carga máxima da viatura, os indicadores de produtividade a adotar (quantidade recolhida e distância media percorrida), as estratégias de procura a implementar e as funcionalidades que o sistema deverá ser capaz de responder, como por exemplo, ser capaz de dado um ponto de partida descobrir quais os circuitos com mais pontos de recolha.

Assim sendo, o presente relatório vem ilustrar em detalhe, com o auxílio de imagens de código ou ilustrativas e tabelas, o trabalho efetuado e expor a abordagem adotada para cumprir com os requisitos anteriormente mencionados.

PRELIMINARES

Antes de iniciar uma leitura mais aprofundada sobre o presente relatório, qualquer leitor deverá ter por base alguns conceitos genéricos, mas fundamentais, para a sua compreensão. Assim, até o leitor mais ingénuo na temática em causa deverá ser capaz de compreender os conceitos discutidos após a leitura deste capítulo.

Comecemos pelo princípio, numa vertente mais técnica o trabalho realizado está assente em alguns princípios fundamentais:

- **Pesquisa Não Informada** – usa apenas as informações disponíveis na definição do problema . Exemplos deste tipo são a pesquisa em profundidade e pesquisa em largura.

- **Pesquisa em Profundidade** – o algoritmo começa num nó raiz e explora tanto quanto possível cada um dos seus ramos, antes de retroceder.

- **Pesquisa em Profundidade Limitada** – o algoritmo começa num nó raiz e explora tanto quanto possível cada um dos seus ramos de acordo com um limite de profundidade máximo estabelecido, antes de retroceder.

- **Pesquisa em Largura** – o algoritmo começa pelo vértice raiz e explora todos os vértices vizinhos. Então, para cada um desses vértices mais próximos, explora-se os vértices vizinhos inexplorados e assim por diante, até que ele encontrar o alvo da pesquisa.

- **Pesquisa Informada** – o algoritmo recebe "dicas" sobre a adequação de diferentes estados. Exemplos deste tipo são a pesquisa A Gulosa e a pesquisa A Estrela.

- **A Gulosa** – o algoritmos tenta resolver o problema fazendo a escolha ótima local em cada fase com a esperança de encontrar um ótimo global.

- **A Estrela –** o algoritmo encontra um caminho no grafo desde o vértice inicial até um vértice final, usando combinações de aproximações heurísticas.

- **Heurística –** é uma técnica projetada para resolver um problema mais rapidamente quando os métodos clássicos são muito lentos ou para encontrar uma solução aproximada quando os métodos clássicos falham em encontrar uma solução exata.

Para finalizar, mencionar apenas que caso procure um aprofundamento dos tópicos supramencionados ou qualquer outro esclarecimento sobre sistemas de representação de conhecimento e raciocínio, deverá ler o capítulo das Referências Bibliográficas que contempla todos os materiais utilizados para o desenvolvimento do projeto.

FORMULAÇÃO DO PROBLEMA

No que diz respeito a formulação do problema, tal como em muitos problemas em ciências da computação, esta pode ser definida formalmente de acordo com cinco componentes, sendo eles :

1. Representação do Estado.
2. Estado Inicial.
3. Relação de Transição entre Estados.
4. Estado Objetivo.
5. Custo da Solução.

No caso de estudo apresentado, os **estados** podem ser considerados como cada ponto de recolha, o **estado inicial** corresponderá a **garagem** que para facilitar o processo de início de percurso terá a mesma localização geográfica de um ponto do dataset, o **estado final** ou objetivo corresponderá ao **depósito** onde mais uma vez será feita a simplificação com um dos pontos do dataset, a **transição entre pontos/nodos** será efetuada com base na lista de nodos adjacentes presente em cada ponto de recolha, o **custo da solução** servirá para avaliar qual a melhor rota a adotar consoante o critério pretendido, o caminho/circuito efetuado poderá ser avaliado através de indicadores como distância média percorrida, quantidade média de resíduos recolhidos e número de nodos visitados.

Existe ainda um fator adicional que limita a circulação entre pontos de recolha, isto é, a capacidade de carga do veículo coletor de lixo, que corresponde a cerca de 15000 litros ou 15 metros cúbicos. Neste sentido, irá ser necessário efetuar desvios ao depósito de forma a conseguir cumprir com o processo de recolha de lixo entre o ponto inicial e o ponto objetivo. Deste modo, foi necessário considerar que o depósito está ligado a qualquer ponto de recolha.

Na globalidade, podemos considerar que os pontos de recolha, garagem e depósito constituem um grafo, o qual poderá ser percorrido e analisado pelos diversos algoritmos de procura implementados.

DESCRIÇÃO DO TRABALHO

De modo a melhor organizar e estruturar o trabalho, o projeto criado foi dividido em diferentes ficheiros PROLOG que agregam em cada qual componentes semelhantes do sistema:

- **BaseConhecimento.pl:** onde estão presentes todos os predicados e conhecimento que sustenta o sistema.
- **Funcionalidades.pl:** neste ficheiro estão implementadas todas as funcionalidades propriamente ditas, implementadas a custa dos algoritmos.
- **RegrasAuxiliares.pl:** este documento agrega todas as regras que serviram de auxílio à implementação das regras principais e algoritmos.
- **Algoritmos.pl:** contém todos os algoritmos necessários para o sistema, ou seja, tanto de pesquisa informada como não informada.

Nesta secção será feita uma abordagem em detalhe de cada um dos ficheiros PROLOG que foram criados bem como do **processamento** efetuado ao dataset fornecido, que convergiu no ficheiro PROLOG "BaseConhecimento.pl".

PROCESSAMENTO DADOS

Numa primeira fase, surge a necessidade de representar o dataset numa base de conhecimento. Contudo, devido as irregularidades do mesmo, foi necessário processar os dados nele contidos.

O dataset original possuía as seguintes informações relativas a cada ponto de recolha : Latitude, Longitude, ID , local ponto de recolha, tipo de resíduo, tipo de contentor, quantidade de contentores no ponto de recolha, capacidade de cada contentor, quantidade total dos contentores do ponto de recolha em litros. Tal como podemos verificar na figura abaixo :

	A	B	C	D	E	F	G	H	I	J	K	L
1	Latitude	Longitude	OBJECTID	PONTO_RECOI	PONTO_RECOLHA_LOCAL	CONTENTOR_RESÍDUO	CONTENTOR_TIPO	CONTENTOR_CAPACIDADE	CONTENTOR_QT		CONTENTOR_TOTAL_LITROS	
2	-9,14331	38,70808	355	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0090	90	1	90		
3	-9,14331	38,70808	356	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0240	240	7	1680		
4	-9,14331	38,70808	357	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0090	90	1	90		
5	-9,14331	38,70808	358	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Papel e Cartão	CV0240	240	6	1440		
6	-9,14331	38,70808	359	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Papel e Cartão	CV0090	90	1	90		
7	-9,14338	38,70808	364	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0240	240	1	240		
8	-9,14338	38,70808	365	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0140	140	6	840		
9	-9,14338	38,70808	366	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0120	120	1	120		
10	-9,14338	38,70808	367	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0090	90	2	180		
11	-9,14338	38,70808	368	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0240	240	1	240		
12	-9,14338	38,70808	369	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0140	140	6	840		
13	-9,14338	38,70808	370	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0120	120	1	120		
14	-9,14338	38,70808	360	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Lixos	CV0090	90	2	180		
15	-9,14338	38,70808	361	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Papel e Cartão	CV0140	140	2	280		
16	-9,14338	38,70808	362	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Papel e Cartão	CV0090	90	1	90		
17	-9,14338	38,70808	363	Misericórdia	15806: R do Alecrim (Impar (27->53)(->): R Ataíde - R Ferragial)	Papel e Cartão	CA0090	90	1	90		

Figura 1: Figura ilustrativa do dataset fornecido

De seguida entramos na etapa de conversão do dataset para um novo ficheiro PROLOG mais relevante e compactado, o que obviamente requereu algum estudo, análise e reflexão prévios. Para a conversão do dataset foi utilizada a linguagem de programação JAVA. O programa desenvolvido é feito por progressivas etapas de refinamento :

1º - faz uma leitura literal do ficheiro CSV fornecido e divide os seus parâmetros por campos.

```
----------DATASET----------
Linhas Lidas : 162
Cabeçalho : [Latitude, Longitude, OBJECTID, PONTO_RECOLHA_FREGUESIA, PONTO_RECOLHA_LOCAL, CONTENTOR_RESÍDUO, CONTENTOR_TIPO, CONTENTOR_CAPACIDADE, CONTENTOR_QT, CONTENTOR_TOTAL_
1ª Linha : [-9.143308809, 38.70807879, 355, Misericórdia, 15805: R do Alecrim (Par (→)(26→30): R Ferragial - R Ataíde), Lixos, CV0090, 90, 1, 90]
1ª Linha : [-9.143308809, 38.70807879, 356, Misericórdia, 15805: R do Alecrim (Par (→)(26→30): R Ferragial - R Ataíde), Lixos, CV0240, 240, 7, 1680]
Ultima Linha: [-9.15158074, 38.70770064, 966, Misericórdia, 21969: R Dom Luis I, 34, Embalagens, CV0120, 90, 1, 90]
```

Figura 2: Exemplo ilustrativo output 1º passo do processamento do dataset

2º - formata as linhas lidas de modo a obter apenas a informação relevante (por exemplo, no campo “local ponto de recolha” apenas considera o nome da rua em questão e o primeiro ponto adjacente caso este exista) e ignora os campos menos relevantes (por exemplo, os campos “tipo de contentor, quantidade de contentores no ponto de recolha e capacidade de cada contentor” são menosprezados pois a informação relevante concentra-se apenas no campo “quantidade total dos contentores do ponto de recolha em litros”).

```
-----Nova Formatação-----
Numero Linhas Formatadas: 161
1° Linha  : [-9.143308809, 38.70807879, 355, R do Alecrim, [R Ferragial], Lixos, 90]
2° Linha  : [-9.143308809, 38.70807879, 356, R do Alecrim, [R Ferragial], Lixos, 1680]
Ultima Linha: [-9.15158074, 38.70770064, 966, R Dom Luís I, [], Embalagens, 90]
```

Figura 3: Exemplo ilustrativo output 2º passo do processamento do dataset

3º - ocorrem os agrupamentos com base na posição geográfica, ou seja, linhas com iguais coordenadas de latitude e longitude serão agrupadas numa só linha, onde os seus vizinhos serão também agrupados e a quantidade de lixo somada. Contudo, há que notar dois fatores : que é feita uma separação das quantidades dos diferentes tipos de lixo; e que o agrupamento dos vizinhos agora tem em conta também a linha seguinte do dataset para além dos pontos de recolha adjacentes, isto permitiu contornar a irregularidade de existir pontos de recolha sem pontos adjacentes/vizinhos.

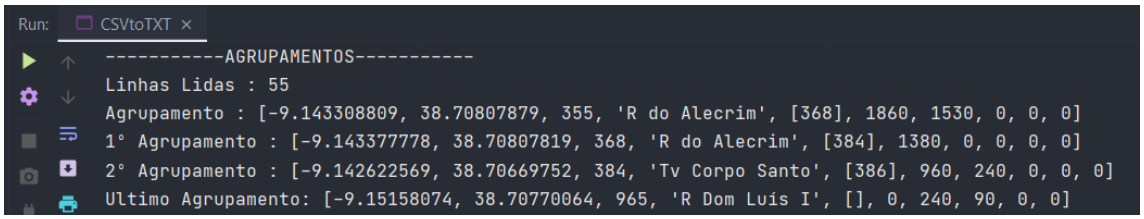

Figura 4: Exemplo ilustrativo output 3º passo do processamento do dataset

4º - a todas as linhas agrupadas são acrescentadas a um predicado ponto e é criado um ficheiro PROLOG com todos os pontos do dataset original agora agrupado por localização geográfica e seletivo por tipo de lixo.

```
1  ponto(-9.143308809, 38.70807879, 355, 'R do Alecrim', [368], 1860, 1530, 0, 0, 0).
2  ponto(-9.143377778, 38.70807819, 368, 'R do Alecrim', [384], 1380, 0, 0, 0, 0).
3  ponto(-9.142622569, 38.70669752, 384, 'Tv Corpo Santo', [386], 960, 240, 0, 0, 0).
4  ponto(-9.142408356, 38.70699663, 386, 'R Bernardino da Costa', [333, 334, 463], 2760, 240, 0, 0, 0).
5  ponto(-9.147044252, 38.70801938, 463, 'R Moeda', [531, 267, 85, 467], 5720, 0, 0, 0, 0).
6  ponto(-9.146144218, 38.70778398, 467, 'Tv Carvalho', [494, 497, 469], 180, 0, 0, 0, 0).
7  ponto(-9.145639847, 38.70788744, 469, 'Tv Carvalho', [467, 469, 474, 843], 1130, 370, 0, 0, 0).
8  ponto(-9.145988636, 38.70816366, 474, 'Tv Carvalho', [467, 469, 474, 843, 482], 140, 0, 0, 0, 0).
```

Figura 5: Exemplo ilustrativo BaseConhecimento.pl - 4º passo do processamento do dataset

De forma a não aborrecer muito o leitor e visto não se tratar do foco do trabalho poupa-se a apresentação em detalhe do código realizado e das estruturas de dados utilizadas. Mencionar apenas que foram usados vários HashMaps quer para mapear ruas ao local quer para mapear ruas ao seu ID, para no final convergir em numa base de conhecimento onde cada registo, aqui denominado por ponto, tem um ID único. Também foram usados ArrayLists para armazenar o conteúdo lido do ficheiro dataset.csv e para armazenar o resultado final do processamento que será então escrito para o ficheiro “BaseConhecimento.pl”.

Notar ainda que este programa funciona tanto para o dataset original como para datasets reduzidos, as imagens apresentadas correspondem a um dataset reduzido e deduzido do original com 160 registos usado para propósitos de teste. De qualquer das formas o ficheiro PROLOG criado terá de ser codificado com UTF-8 de forma a ultrapassar as irregularidades levantadas pelos acentos nos nomes das ruas.

BASE DE CONHECIMENTO

Tendo efetuado o processamento do dataset temos agora em mãos um ficheiro PROLOG que representará a base de conhecimento para o sistema de recolhas de resíduos urbanos, cujo excerto apresentado de seguida serve para facilitar a descrição dos campos nele contido.

```
ponto(-9.143308809, 38.70807879, 355, 'R do Alecrim', [906, 364], 1860, 1530, 0, 0, 0).
ponto(-9.143377778, 38.70807819, 364, 'R do Alecrim', [371], 2760, 460, 0, 0, 0).
ponto(-9.143481807, 38.70730262, 371, 'R do Alecrim', [906, 377], 1320, 380, 0, 0, 0).
ponto(-9.142550987, 38.70732868, 377, 'R Corpo Santo', [333, 334, 342, 381], 2480, 0, 0, 0, 0).
ponto(-9.142765961, 38.70708361, 381, 'Tv Corpo Santo', [377, 383], 400, 0, 0, 0, 0).
ponto(-9.142622569, 38.70669752, 383, 'Tv Corpo Santo', [386, 391], 1920, 240, 0, 0, 0).
ponto(-9.142408356, 38.70699663, 386, 'R Bernardino da Costa', [333, 334, 342, 391], 2760, 240, 0, 0, 0).
ponto(-9.143050155, 38.70685596, 391, 'R Bernardino da Costa', [381, 383, 397], 3560, 0, 0, 0, 0).
ponto(-9.151251124, 38.70877545, 397, 'Lg Conde-Barão', [418, 423, 426, 429, 433, 437, 921, 406], 6940, 0, 0, 0, 0).
ponto(-9.151789464, 38.70863563, 406, 'Lg Conde-Barão', [447, 448, 411], 3020, 0, 0, 0, 0).
ponto(-9.152018979, 38.70860661, 411, 'Lg Conde-Barão', [417], 1280, 0, 0, 0, 0).
ponto(-9.149105527, 38.70907338, 417, 'Tv Marquês de Sampaio', [418, 423, 426, 429, 433, 437, 921], 240, 0, 0, 0, 0).
ponto(-9.147649761, 38.70856359, 418, 'R da Boavista', [504, 510, 516, 521, 527, 534, 543, 549, 553, 563, 267, 85, 423], 3500, 0, 0, 0, 0).
ponto(-9.14857178, 38.70872672, 423, 'R da Boavista', [417, 426], 1410, 0, 0, 0, 0).
ponto(-9.149113446, 38.7088211, 426, 'R da Boavista', [429], 850, 0, 0, 0, 0).
ponto(-9.149493546, 38.70887183, 429, 'R da Boavista', [417, 433], 1160, 0, 0, 0, 0).
ponto(-9.150183884, 38.70891086, 433, 'R da Boavista', [437], 5190, 0, 0, 0, 0).
ponto(-9.150793117, 38.70890555, 437, 'R da Boavista', [457, 459, 944, 440], 490, 0, 0, 0, 0).
ponto(-9.152060348, 38.70828196, 440, 'Tv do Cais do Tojo', [441, 443], 140, 0, 0, 0, 0).
ponto(-9.152472114, 38.70813424, 441, 'R Cais do Tojo', [443], 2040, 0, 0, 0, 0).
ponto(-9.151783576, 38.70822132, 443, 'R Cais do Tojo', [440, 447], 1110, 0, 0, 0, 0).
ponto(-9.151544743, 38.70840356, 447, 'Bqr do Duro', [441, 443, 448], 1200, 0, 0, 0, 0).
ponto(-9.151388528, 38.70792751, 448, 'Bqr do Duro', [441, 443, 452], 1330, 0, 0, 0, 0).
ponto(-9.147513949, 38.71033931, 452, 'Tv Santa Catarina', [455], 480, 0, 0, 240, 0).
ponto(-9.148566019, 38.70994283, 455, 'R Ferreiros a Santa Catarina', [457], 1440, 0, 0, 0, 0).
ponto(-9.150464329, 38.70680059, 457, 'R Instituto Industrial', [351, 354, 459], 2060, 0, 0, 0, 0).
ponto(-9.15054864, 38.70788079, 459, 'R Instituto Industrial', [476, 478, 479, 481, 965, 880, 461], 5660, 0, 0, 0, 0).
ponto(-9.148518124, 38.70980813, 461, 'R Santa Catarina', [463], 1920, 0, 0, 0, 0).
ponto(-9.147044252, 38.70801938, 463, 'R Moeda', [504, 510, 516, 521, 527, 534, 543, 549, 553, 563, 267, 85, 467], 5720, 0, 0, 0, 0).
```

Figura 6: Excerto exemplo ficheiro BaseConhecimento.pl

Como podemos ver na imagem anterior, para representar o conhecimento associado a cada ponto de recolha do dataset usou-se o predicado ponto, onde estão contidos os campos : latitude, longitude, ID ponto de recolha, nome da rua, lista de pontos adjacentes, quantidade de lixo do tipo não seletivo ou normal, quantidade de lixo seletivo do tipo papel e cartão, quantidade de lixo seletivo do tipo embalagens, quantidade de lixo seletivo do tipo orgânico e quantidade de lixo seletivo do tipo vidro .

Tendo enunciado os elementos constituintes do registo do ponto de recolha, falta agora apresentar as funcionalidade inerentes a cada campo, assim sendo passo a sua descrição :

- A latitude e a longitude funcionam em conjunto para concretizar as coordenadas geográficas do ponto de recolha, tendo como principal função servir de base para futuros cálculos de distância entre pontos do dataset.
- O ID representa o ponto de recolha, será o seu identificador inequívoco.
- O nome da rua serve simplesmente para efeitos conhecimento e associação do ponto no nosso programa a rua onde este se encontra mundo real do ponto em causa.
- A lista de nodos vizinhos serve definir quais os pontos que podem ser acedidos a partir do nodo onde nos encontramos no momento, é bastante para o funcionamento e execução dos algoritmos.
- Finalmente as diversas quantidades de lixo separadas conforme o seu tipo servem novamente para a execução dos algoritmos.

Por último, visto se tratar de um ficheiro PROLOG bastará fazer “:- include ('BaseConhecimento.pl').” para conseguirmos aceder ao conhecimento presente na base de conhecimento no ficheiro PROLOG mais adequado, neste caso será em “Funcionalidades.pl”.

ALGORITMOS

Encontramo-nos agora numa etapa posterior, a algoritmia, aqui serão abordados com detalhe os algoritmos implementados, quer de pesquisa não informada (DF, DF-Limitado, BF) como informada (Gulosa, A*), presentes no ficheiro “Algoritmos.pl”, capazes de recomendar circuitos de recolha após executarem a travessia do grafo representado pelos pontos da base de conhecimento.

PESQUISA NÃO INFORMADA

A primeira estratégia de pesquisa retratada será a pesquisa não informada (cega), nestas estratégias usam-se apenas as informações disponíveis na definição do problema. Dentro desta categoria estão inseridos os algoritmos de Primeiro em Largura e Primeiro em Profundidade, que em seguida serão analisados.

Pesquisa em Profundidade (DF)

A estratégia passa por expandir sempre um dos nós mais profundos da árvore, se não for possível recuar e tentar em alternativa outros caminhos. Requer pouca memória e é bom para problemas com muitas soluções. Em contrapartida, não pode ser usado para árvores com profundidade infinita, pode ficar presa em ramos errados e não garante o caminho mais curto ou com menos passos em primeiro lugar.

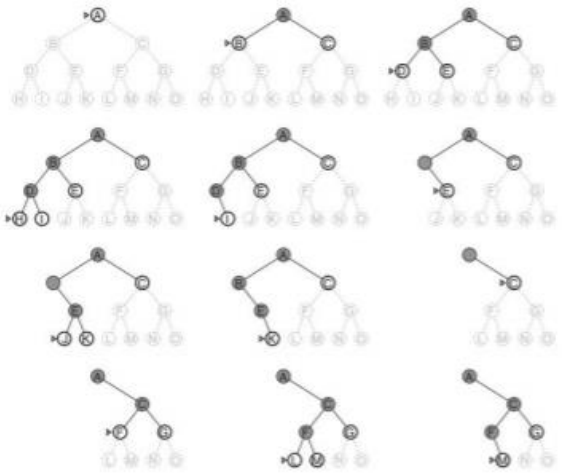

Figura 7: Exemplo ilustrativo Pesquisa em Profundidade

Este algoritmo aplica-se para problemas com várias soluções, podendo ser bem mais rápida do que a procura em largura. No entanto, deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos.

Passemos agora ao código desenvolvido, que se encontra no ficheiro PROLOG “Algoritmos.pl”.

A regra **resolveDFFinal** é a responsável pela implementação do algoritmo de Depth First, como podemos ver é multiestados uma vez que permite a inserção de um ponto inicial (I) e um ponto destino (F), o terceiro campo corresponde ao caminho solução encontrado, o quarto campo devolve todos as quantidades em litros de lixo recolhido separadas por tipo de lixo, nomeadamente : quantidade lixo normal / quantidade papel / quantidade embalagens / quantidade vidro / quantidade lixo orgânico. De seguida apresenta ainda métricas como os Kms percorridos, o número de nodos visitados durante o percurso, a quantidade Total de lixo recolhido e o numero de idas ao deposito.

```
% Profundidade (DFS - Depth-First Search)
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS

resolveDFFinal(I, F, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito) :-
                                                resolveDFF(I, F, [I],SolucaoAux),
                                                inverso([I|SolucaoAux],NovoInv),
                                                deposito(NovoInv,Solucao, DepCamiao,IdasDeposito),
                                                comprimento(Solucao, L),
                                                Visitas is L,
                                                kilometros(Solucao,Kms),
                                                qntLixos(Solucao,CL,CP,CE,CV,CO),
                                                QntTotal is CL+CP+CE+CV+CO.     % Quantidade total de lixo - todo tipo.
```

Figura 8: Regra resolveDFFinal que implementa DF

Esta função recorre a outra função muito importante chamada **resolveDFF** que é a responsável por apresentar-lhe a solução obtida e onde de facto ocorre o esforço algorítmico, de resto a função resolveDFFinal apenas calcula com base no caminho retornado pela função resolveDFF todos os campos extra para além do percurso como os quilómetros percorridos, as idas ao deposito , numero de pontos de recolha visitados e as quantidades lixo recolhidas, através de funções auxiliares presentes no ficheiro “RegrasAuxiliares.pl”.

De seguida segue-se a regra resolveDFF, que consiste praticamente no caso de paragem, ou seja, atingimos o destino e o caso recursivo, ou seja, todas as verificações necessárias até chegarmos ao fim, nomeadamente a atenção aos nodos visitados e a iteração conforme a lista de nodos adjacentes ao atual. A noção de adjacente está implementada nas “RegrasAuxiliares.pl” mas em concreto apenas estipula que o grafo é unidirecional e que o próximo nodo tem de ser adjacente ao nodo atual, o predicado nao e membro também se encontram definidos nas “RegrasAuxiliares.pl” e garantem que o próximo nodo a ser visitado ainda não foi visitado.

```
resolveDFF(Estado, Final,_,[]) :- Estado == Final, !.%o prolog constroi a solução fim para o incio é por
resolveDFF(Estado, Final, Historico,[Move|Solucao]) :-
                                                adjacente(Estado,Move),
                                                nao(membro(Move,Historico)),
                                                resolveDFF(Move,Final,[Move|Historico],Solucao).
```

Figura 9: Regra resolveDFF

Pesquisa em Profundidade Limitada (DF-Limitado)

Este método de procura funciona de maneira análoga a pesquisa em profundidade, só que com limite de profundidade l, ie, nós em profundidade I não têm sucessores.

Um dos problemas da pesquisa Primeiro em Profundidade prende-se com a incapacidade desta lidar com caminhos infinitos. A Pesquisa em Profundidade com limite de profundidade procura evitar este problema fixando o nível máximo de procura.

Passemos ao código desenvolvido, presente no ficheiro PROLOG "Algoritmos.pl". Como já foi mencionado, o raciocínio para a regra **resolveDFLimite** é idêntico a função implementada em Profundidade, a única diferença consiste na adição de um limite a profundidade. Caso a função não tenha capacidade de atingir uma solução no limite de profundidade estipulado retornará False.

```
% Busca Iterativa Limitada em Profundidade
% LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS

resolveDFLimite(I,F,Solucao,Lim, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito)  :- resolveDFFLimite(I, F, [I],SolucaoAux, Lim),
                                                                                         inverso([I|SolucaoAux],NovoInv),
                                                                                         deposito(NovoInv,Solucao, DepCamiao,IdasDeposito),
                                                                                         comprimento(Solucao, L),
                                                                                         Visitas is L,
                                                                                         kilometros(Solucao,Kms),
                                                                                         qntLixos(Solucao,CL,CP,CE,CV,CO),
                                                                                         QntTotal is CL+CP+CE+CV+CO.     % Quantidade total de lixo - todo tipo.
```

Figura 10: Regra resolveDFLimite que implementa DF-Limitado

Neste caso a regra auxiliar responsável pelo passo recursivo e pela solução atingida é a **resolveDFFLimite**, aqui podemos ver mais concretamente como de facto se processa o mecanismo de limitação da profundidade. Mais uma vez, todos os mecanismos auxiliares encontra-se no ficheiro PROLOG "RegrasAuxiliares.pl".

```
resolveDFFLimite(Estado, Final,_,[],_) :- Estado == Final, !. % o prolog constroi a solução fim para o incio é por is
resolveDFFLimite(Estado, Final, Historico,[Move|Solucao],Lim) :- Lim > 1, % ja conta inicial
                                                                 adjacente(Estado,Move),
                                                                 Lim1 is Lim -1,
                                                                 nao(membro(Move,Historico)),
                                                                 resolveDFFLimite(Move,Final,[Move|Historico],Solucao, Lim1).
```

Figura 11: Regra resolveDFFLimite

Pesquisa em Largura (BF)

A estratégia passa por a partir de um nó serem explorados todos os nós adjacentes e só em seguida explorados os nós acessíveis através dos adjacentes (ou seja, nível seguinte) e assim sucessivamente. A pesquisa é muito sistemática e garante que o caminho com menos passos é encontrado em primeiro lugar. Em contrapartida, a estrutura de dados requerida é pesada pois é preciso armazenar todos os caminho ainda por expandir, o que ocupa muito espaço.

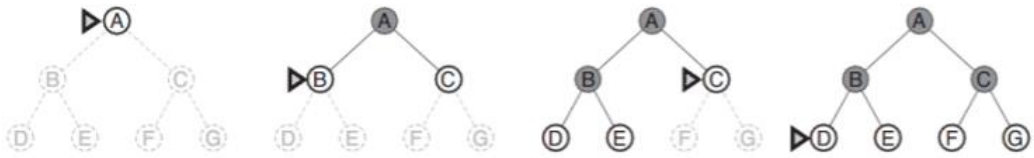

Figura 12: Exemplo ilustrativo Pesquisa em Largura

Este algoritmo deve aplicar-se apenas a pequenos problemas devido ao seu custo em memória, contudo existem várias aplicações deste algoritmo, tais como : achar componentes conectados de um grafo, achar o menor caminho entre um nó raiz e outros nós do grafo e testar bipartição em grafos.

Voltamos agora a nossa atenção para a explicação do código desenvolvido, ou seja, a implementação da pesquisa em Largura ou BF (Breadth First).

A regra responsável pelo algoritmo de Largura Primeiro é a **resolveBF**, tal como para a pesquisa em profundidade trata-se também de uma função multiestados visto que permite inserir um valor de inicio de percurso (I) e de destino final (F). Os restantes campos da regra permitem obter respostas como : qual o caminho encontrado, as quantidades de lixo de cada tipo recolhidas, os Kms percorridos, o número de pontos de recolha visitados, a quantidade total de lixo coletado em litros ou metros cúbicos e o número de idas ao depósito, respetivamente. Todas as regras auxiliares encontram-se no ficheiro PROLOG "RegrasAuxiliares.pl", tal e qual como sucedeu no exemplo da profundidade.

```
%---------------------------------
% Largura (BFS - Breadth-First Search)

resolveBF(I,F,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito)  :- resolveBFFinal(F, [[I]], SolucaoAux),
                                                    inverso(SolucaoAux,NovoInv),
                                                    deposito(NovoInv,Solucao, DepCamiao,IdasDeposito),
                                                    comprimento(Solucao, L),
                                                    Visitas is L,
                                                    kilometros(Solucao,Kms),
                                                    qntLixos(Solucao,CL,CP,CE,CV,CO),
                                                    QntTotal is CL+CP+CE+CV+CO.     % Quantidade total de lixo - todo tipo.
```

Figura 13: Regra resolveBF que implementa BF

As diferenças entre pesquisas não informadas surge pois na regra auxiliar **resolveBFFinal**.

A função está definida através de um caso de paragem, onde quando o objetivo é atingido (F) o caminho (path) é instanciado. No caso recursivo não acontece nada de especial, simplesmente tem de se manter uma fila de listas de visitados, ao passo que na procura em profundidade apenas tínhamos uma só lista de visitados, apenas temos de ter alguns cuidados de fazer add e pop corretamente para manter uma ordem FIFO.

A noção de adjacente2 está implementada nas "RegrasAuxiliares.pl" mas em concreto apenas estipula que o grafo tem ligações bidirecional, ou seja que se X é vizinho de Y, também Y é vizinho de X, e desta forma que o próximo nodo tem de ser adjacente ao nodo atual, o predicado nao e membro também se encontram definidos nas "RegrasAuxiliares.pl" e garantem ,em conjunto, que o próximo nodo a ser visitado ainda não foi visitado. A regra inverso serve para calcular o inverso de uma lista e encontra-se também nas "RegrasAuxiliares.pl". São usados ainda as definições do PROLOG append e maplist, bem como a regra AdicionaQueue para garantir o correto funcionamento da procura em largura.

Na figura a seguir podemos ver a implementação da função resolveBFFinal e todas as restantes regras discutidas.

```
resolveBFFinal( F, [[F|Visitados] |_], Path) :- inverso([F|Visitados], Path).
resolveBFFinal( F, [Visitados|Resto], Path) :-
    Visitados = [Atual|_],
    Atual \== F,
    findall( X,
        ( adjacente2(Atual, X), nao(membro(X, Visitados)) ),
        [T|Extensao]),
    maplist( adicionaQueue(Visitados), [T|Extensao], VisitadosExtra),
    append( Resto, VisitadosExtra, NovaFila),
    resolveBFFinal( F, NovaFila, Path).

adicionaQueue( A, B, [B|A]).
```

Figura 14: Regra resolveBFFinal

PESQUISA INFORMADA

A segunda e última estratégia de pesquisa retratada é a pesquisa informada (heurística), nestas estratégias dá-se ao algoritmo "dicas" sobre a adequação de diferentes estados para evitar que o algoritmo fique "perdido". Dentro desta categoria estão inseridos os algoritmos de Pesquisa Gulosa e o Algoritmo A* que serão de seguida explicados.

Heurística dos Algoritmos de Procura Informada

Antes de entrar em pormenores sobre cada algoritmo, devemos saber o que são heurísticas, qual o seu propósito, quais foram as heurísticas adotadas e o porque da sua escolha. Primeiramente uma heurística é nada mais nada menos que um método/processo para resolução de resolução de problemas. Numa procura onde o universo é totalmente conhecido, a heurística será realizada através da atribuição de números, senão será realizada através da aplicação de regras.

Cada problema tem uma heurística especifica para o resolver, não havendo uma heurística genérica que sirva para resolver todo o tipo de problemas. No caso de estudo a heurística escolhida foi a distância, dado termos acesso as posições geográficas de cada ponto de recolha pareceu uma opção adequada. Como forma a minimizar o custo de ida e volta ao depósito quando um camião se encontra cheio, a heurística escolhida tem como critério a distância do ponto de recolha ao depósito.

A GULOSA

A estratégia passa por expandir o nó que parece estar mais perto da solução, através de h(n) = custo estimado do caminho mais curto do estado n para o objetivo (função heurística). Pode retornar uma solução ou falhar. A complexidade temporal e espacial dependem da heurística.

Relativamente ao código desenvolvido comecemos pela regra **resolve_gulosa** que é responsável por dado um nodo inicial atingir um nodo objetivo, definido como goal(965), o nodo objetivo definido será igual a um ponto de recolha do dataset, a escolha do ponto 965 foi arbitrária e serve como exemplo demonstrativo. Como já foi discutido nos restantes algoritmos a regra devolve o caminho encontrado e os atributos desse caminho, nomeadamente as quantidades recolhidas dos diferentes tipo de lixo, os Kms percorridos, o número de pontos de recolha visitados, a quantidade total em litros de lixo recolhido e o número de idas ao depósito.

```
% Gulosa

resolve_gulosa(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito) :-
    estima(Nodo, Estima),
    agulosa([[Nodo]/0/Estima], InvCaminho/Custo/_),
    deposito(InvCaminho,Solucao, DepCamiao,IdasDeposito),
    comprimento(Solucao, L),
    Visitas is L,
    kilometros(Solucao,Kms),
    qntLixos(Solucao,CL,CP,CE,CV,CO),
    QntTotal is CL+CP+CE+CV+CO.     % Quantidade total de lixo - todo tipo.
```

Figura 15 : Regra resolve_gulosa que implementa pesquisa gulosa

A regra resolve_gulosa para além de recorrer as demais funções auxiliares, usa também uma regra auxiliar fundamental chamada **agulosa** que tal em casos anteriores trata da pesquisa em si e está definida pelos casos de paragem e recursivo definidos na seguinte figura.

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

agulosa(Caminhos, SolucaoCaminho) :-
    obtem_melhor_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos), % retira caminho da lista
    expande_gulosa(MelhorCaminho, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovosCaminhos),
    agulosa(NovosCaminhos, SolucaoCaminho).
```

Figura 16: Regra agulosa

Esta, por sua vez, chama duas outras importantes regras, a regra **obtem_melhor_g** e a regra **expande_gulosa**, que são responsáveis por expandir o algoritmo de pesquisa gulosa permite a obtenção do circuito. Tal como podemos verificar na seguinte figura.

```
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, adjacente3(Caminho, NovoCaminho), ExpCaminhos).
```

Figura 17: Regras obtem_melhor_g e expande_gulosa

A regra **adjacente3**, utilizada na expande gulosa, usa a já referida regra de adjacente2 que tem em conta as ligações de um grafo com sentido bidirecional e acrescenta ainda a noção de custos ao caminho através das regras auxiliares estima e distância. A regra estima serve para calcular a distância entre o nodo seguinte e o depósito, enquanto que a regra distância serve para calcular a distância entre o nodo e o próximo nodo, ou por outras palavras o passo do custo. Ambas as regras encontra-se no ficheiro PROLOG “RegrasAuxiliares.pl” e são fundamentais para representar a heurística definida para a procura gulosa. Para melhor entendimento segue na figura abaixo a implementação do código para a regra adjacente3.

```
% evitamosCiclos
adjacente3([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Estimativa) :-
    adjacente2(Nodo, ProxNodo),
    distancia(Nodo, ProxNodo, PassoCusto),
    nao(membro(ProxNodo,Caminho)),
    NovoCusto is Custo + PassoCusto,
    estima(ProxNodo,Estimativa).
```

Figura 18: Regra adjacente3

A Estrela (A *)

A estratégia passa por evitar expandir caminhos que são caros, o algoritmo A* combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. Ou seja, usa a função :

$$f(n) = g(n) + h(n) \text{ , sendo}$$

g(n) = custo total, até agora, para chegar ao estado n (custo do percurso)

h(n) = custo estimado para chegar ao objetivo (não deve sobrestimar o custo para chegar à solução (heurística))

f(n)= custo estimado da solução mais barata que passa pelo nó n

Relativamente ao código desenvolvido podemos verificar na imagem seguinte que a regra responsável pela implementação do método de procura a estrela é a regra **resolve_aestrela**. O seu funcionamento depende do input do utilizador no campo nodo como forma a definir o nodo inicial, ou por outras palavras o início do circuito e também tem de estar a partida definido um goal(IDnodo) no ficheiro RegrasAuxiliares de modo a definir o destino final do circuito. As métricas resultantes da invocação da regra mantem-se iguais as das pesquisas anteriores, ou seja, é possível apreender a solução ótima e o seu respetivo custo, indicadores de produtividade como as quantidades de lixo por tipo ou globalmente recolhidas e número de Kms percorridos, bem como quantas idas foram necessárias ao depósito e quantos pontos de recolha foram visitados.

```
% A*

resolve_aestrela(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito) :-
    estima(Nodo, Estima),
    aestrela([[Nodo]/0/Estima], InvCaminho/Custo/_),
    deposito(InvCaminho,Solucao, DepCamiao,IdasDeposito),
    comprimento(Solucao, L),
    Visitas is L,
    kilometros(Solucao,Kms),
    qntLixos(Solucao,CL,CP,CE,CV,CO),
    QntTotal is CL+CP+CE+CV+CO.     % Quantidade total de lixo - todo tipo.
```

Figura 19: Regra resolve_aestrela que implementa pesquisa a estrela

A função responsável pela descoberta do caminho em si é a regra **aestrela**, que em si figura um caso de paragem e um caso recursivo como podemos ver na figura seguinte. Além disso, esta regra usa as regras **expande_aestrela** para expandir o algoritmo através da regra adjacente3 e a regra **obtem_melhor** para ter em conta o custo + a estimativa de forma a escolher a melhor opção na seguinte iteração.

```
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, NovosCaminhos),
        aestrela(NovosCaminhos, SolucaoCaminho).
```

Figura 20: Regra aestrela

Tal como na pesquisa gulosa são usadas funcionalidades auxiliares como : a regra estima para estimar a distância do nodo ao deposito, distância para apreender qual a distância/custo do passo de um nodo para o próximo. Estas funcionalidades surgem em concreto na regra adjacente3 abordada já para a pesquisa gulosa e tem o mesmo propósito na pesquisa estrela que é implementar a heurística que contabilize a distância ao depósito. Para entender a implementação da regra adjacente3 deve consultar novamente a figura 18. Na figura 21 vemos então a aplicação do adjacente3 na funcionalidade expande_estrela.

```
% custo + estimativa
% antes era so custo
obtem_melhor([Caminho],Caminho) :- !.
obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],MelhorCaminho) :-
        (Est1 + Custo1) =< (Est2 + Custo2), !,
        obtem_melhor([Caminho1/Custo1/Est1|Caminhos],MelhorCaminho).
obtem_melhor([_|Caminhos],MelhorCaminho) :-
        obtem_melhor(Caminhos,MelhorCaminho).

expande_aestrela(Caminho, ExpCaminhos) :-
    findall(NovoCaminho, adjacente3(Caminho, NovoCaminho), ExpCaminhos).
```

Figura 21: Regras obtem_melhor e expande_estrela

Posteriormente serão apresentados testes em swi_prolog dos algoritmos implementados onde consegue-se de facto ver a eficiência da aplicação do algoritmo a estrela, onde a primeira solução apresenta é de facto a melhor e a restante soluções alternativas são gradualmente piores em termos de distância / custo. O que era de esperar de acordo com a heurística implementada e veio a comprovar o correto funcionamento do algoritmo.

COMPLEXIDADE DOS ALGORITMOS

Focando o assunto numa vertente voltada para a complexidade dos algoritmos anteriormente apresentados (BF, DF, DF-LIM, GULOSA, A*), temos de ter em conta que a complexidade irá depender de caso para caso dado que depende de fatores como o tamanho do grafo e o espaço dos estados. Deste modo, a complexidade dos algoritmos é medida em termos de :

b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de pesquisa.

d: a profundidade da melhor solução.

m: a máxima profundidade do espaço de estados.

l: a profundidade limite da pesquisa.

Assim sendo, podemos analisar o desempenho dos algoritmos de pesquisa com base nos seguintes **critérios**:

- Completude: Está garantido que encontra a solução?
- Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
- Complexidade no Espaço: Quanta memória necessita para fazer a pesquisa?
- Otimalidade: Encontra a melhor solução?

Critério	Largura	Profundidade	Profundidade-Limite
Completa	Sim, se b finito.	Não.	Não.
Tempo	O(b^d+1), exponencial em d.	O(b^m), mau se m > d.	O(b^i)
Espaço	O(b^d+1), guarda nó em memória.	O(bm), espaço linear.	O(bi)
Ótimo	Sim, se custos escalonadas forem iguais.	Não, devolve em principio 1º solução encontrada.	Não, devolve em principio 1º solução encontrada.

Tabela 1: Comparação Estratégias Pesquisa Não Informada

Critério	Gulosa	A *
Completa	Não, pode entrar em ciclos. Suscetível a falsos começos.	Sim.
Tempo	Pior caso: O(b^m). Melhor caso: O(bd).	Número de nodos com g(n)+h(n) ≤ C*
Espaço	Pior caso: O(b^m). Melhor caso: O(bd).	Número de nodos com g(n)+h(n) ≤ C*
Ótimo	Não, não encontra sempre a solução ótima.	Sim, se a heurística for admissível.

Tabela 2: Comparação Estratégias Pesquisa Informada

FUNCIONALIDADES

O ficheiro PROLOG "Funcionalidades.pl" serviu, tal como o nome sugere, para representar regras capazes de cumprir com as funcionalidades propostas a desenvolver no enunciado do trabalho prático individual. Apesar disso, esta classe não é a única responsável por cumprir com os requisitos estipulados, precisando portanto de funcionar em conjunto com o ficheiro "Algoritmos.pl".

A elaboração do caso prático surge com o intuito de permitir, tendo em conta a localização de um ponto de partida (uma garagem):

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

Em adição, os **indicadores de produtividade** a utilizar serão:

- A **quantidade recolhida**: quantidade de resíduos recolhidos durante o circuito;
- A **distância média** percorrida entre pontos de recolha.

Relativamente à **primeira funcionalidade**, todos os algoritmos de pesquisa implementados permitem ter em conta circuitos de recolha seletiva. A implementação baseou-se na ideia de para cada percurso conseguir ser seletivo seria necessário diferenciar as **quantidades de lixo recolhidas** para cada tipo de lixo existente, esta tarefa revelou-se fácil na medida que no processamento dos dados do dataset cada ponto de recolha já apresenta cinco quantidades de lixo diferentes, uma para cada tipo.

Para a completude desta funcionalidade teve-se ainda em conta que se o percurso for, por exemplo, seletivo do tipo vidro e quisermos ir de um ponto A inicial para um ponto B final, muito provavelmente teremos de passar por pontos intermédios que estabelecem ligação entre A e B mas que podem conter ou não resíduos do tipo vidro. Assim sendo, visto que os pontos intermédios são fundamentais para chegarmos do ponto inicial ao ponto de destino, todos os pontos do percurso entre A e B serão tidos em conta no cálculo da **distância média percorrida**. Este exemplo para o caso de um percurso seletivo do tipo vidro pode ser generalizado para todos os outros tipos de lixo.

Na secção dos algoritmos, podemos já ver que estes tinham em conta a noção de quantidade de lixo seletiva. Obviamente que quando pretendermos obter o percurso efetuado entre um ponto A e um ponto B do tipo seletivo vidro, o único campo relevante será a quantidade de lixo vidro recolhido (CV). Não obstante, os restantes campos de quantidade de lixo estarão preenchidos na mesma pois o algoritmo detetará outros tipos de lixo nos ponto de recolha visitados.

Para além de todos os algoritmos implementados (BF, DF, DF-Lim, Gulosa e A*) que mais uma vez cumprem com este requisito na classe “Algoritmos.pl”, a título de exemplo da possibilidade de apresentar ao utilizador apenas o valor da quantidade de lixo do tipo seletivo selecionado, foi implementada na classe “Funcionalidades.pl” uma abstração dos algoritmos de procura não informada e informada.

As figuras a seguir demonstram o código desenvolvido mas a única diferença é o novo parâmetro “Tipo” que irá permitir mostrar apenas a quantidade do lixo do Tipo pretendido, podendo este campo ser iniciado com : “lixo”, “papel”, “embalagens”, “vidro”, “organicos” ou “indiferenciado”.

```
% Profundidade (DFS - Depth-First Search)
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS
% Limitado por Tipo de lixo

% Teste Melhor e Mais Detalhado : resolveDFFinal(355, 921, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).

resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='lixo',
                                    resolveDFFinal(I, F, Solucao, QntLixo/CP/CE/CV/CO, Kms,NrVisitas,QntTotal, IdasDeposito).
resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='papel',
                                    resolveDFFinal(I, F, Solucao,CL/QntLixo/CE/CV/CO, Kms,NrVisitas,QntTotal, IdasDeposito).
resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='embalagens',
                                    resolveDFFinal(I, F, Solucao,CL/CP/QntLixo/CV/CO, Kms,NrVisitas,QntTotal, IdasDeposito).
resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='vidro',
                                    resolveDFFinal(I, F, Solucao,CL/CP/CE/QntLixo/CO, Kms,NrVisitas,QntTotal, IdasDeposito).
resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='organicos',
                                    resolveDFFinal(I, F, Solucao,CL/CP/CE/CV/QntLixo, Kms,NrVisitas,QntTotal, IdasDeposito).
resolveDFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, NrVisitas, IdasDeposito) :-
                                    Tipo=='indiferenciado',
                                    resolveDFFinal(I, F, Solucao,CL/CP/CE/CV/CO, Kms,NrVisitas,QntLixo, IdasDeposito).
```

Figura 22: DF Seletivo

```
% Busca Iterativa Limitada em Profundidade
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS
% Limitado por Tipo de lixo

% Teste Melhor e Mais Detalhado : resolveDFLimite(355,921,Solucao,20, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito).

resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='lixo',
                                    resolveDFLimite(I, F, Solucao, Lim, QntLixo/CP/CE/CV/CO, Kms, NrVisitas, QntTotal, IdasDeposito).
resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='papel',
                                    resolveDFLimite(I, F, Solucao, Lim, CL/QntLixo/CE/CV/CO, Kms, NrVisitas, QntTotal, IdasDeposito).
resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='embalagens',
                                    resolveDFLimite(I, F, Solucao, Lim, CL/CP/QntLixo/CV/CO, Kms, NrVisitas, QntTotal, IdasDeposito).
resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='vidro',
                                    resolveDFLimite(I, F, Solucao, Lim, CL/CP/CE/QntLixo/CO, Kms, NrVisitas, QntTotal, IdasDeposito).
resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='organicos',
                                    resolveDFLimite(I, F, Solucao, Lim, CL/CP/CE/CV/QntLixo, Kms, NrVisitas, QntTotal, IdasDeposito).
resolveDFLimiteSeletivo(I, F, Lim, Tipo, Solucao, QntLixo, Kms,  NrVisitas, IdasDeposito) :-
                                    Tipo=='indiferenciado',
                                    resolveDFLimite(I, F, Solucao, Lim, CL/CP/CE/CV/CO, Kms, NrVisitas,QntLixo, IdasDeposito).
```

Figura 23: DF-Limitado Seletivo

```
% Largura (BFS - Breadth-First Search)
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS
% Limitado por Tipo de lixo

% Teste : resolveBFSeletivo(651, 662,lixo, Solucao, QntLixo, Kms, Visitas).
% Teste Melhor e Mais Detalhado : resolveBF(651,662,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal).

resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='lixo',
                                    resolveBF(I,F,Solucao, QntLixo/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='papel',
                                    resolveBF(I,F,Solucao, CL/QntLixo/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='embalagens',
                                    resolveBF(I,F,Solucao, CL/CP/QntLixo/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='vidro',
                                    resolveBF(I,F,Solucao, CL/CP/CE/QntLixo/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='organicos',
                                    resolveBF(I,F,Solucao, CL/CP/CE/CV/QntLixo, Kms,Visitas, QntTotal, IdasDeposito).
resolveBFSeletivo(I, F,Tipo, Solucao, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='indiferenciado',
                                    resolveBF(I,F,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntLixo, IdasDeposito).
```

Figura 24: BF Seletivo

```
% gulosa
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS
% Limitado por Tipo de lixo
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='lixo',
                                    resolve_gulosa(Nodo, Solucao/Custo, QntLixo/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='papel',
                                    resolve_gulosa(Nodo, Solucao/Custo, CL/QntLixo/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='embalagens',
                                    resolve_gulosa(Nodo, Solucao/Custo, CL/CP/QntLixo/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='vidro',
                                    resolve_gulosa(Nodo, Solucao/Custo, CL/CP/CE/QntLixo/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='organicos',
                                    resolve_gulosa(Nodo, Solucao/Custo, CL/CP/CE/CV/QntLixo, Kms,Visitas, QntTotal, IdasDeposito).
resolveGulosaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms, Visitas, IdasDeposito) :-
                                    Tipo=='indiferenciado',
                                    resolve_gulosa(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntLixo, IdasDeposito).
```

Figura 25: Gulosa Seletivo

```
% A*
% Divisao por lixo: LIXO   PAPEL   EMBALAGENS   VIDRO   ORGANICOS
% Limitado por Tipo de lixo
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='lixo',
                                    resolve_aestrela(Nodo, Solucao/Custo, QntLixo/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='papel',
                                    resolve_aestrela(Nodo, Solucao/Custo, CL/QntLixo/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='embalagens',
                                    resolve_aestrela(Nodo, Solucao/Custo, CL/CP/QntLixo/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='vidro',
                                    resolve_aestrela(Nodo, Solucao/Custo, CL/CP/CE/QntLixo/CO, Kms,Visitas, QntTotal, IdasDeposito).
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='organicos',
                                    resolve_aestrela(Nodo, Solucao/Custo, CL/CP/CE/CV/QntLixo, Kms,Visitas, QntTotal, IdasDeposito).
resolveAEstrelaSeletivo(Nodo, Tipo, Solucao/Custo, QntLixo, Kms,Visitas, IdasDeposito) :-
                                    Tipo=='indiferenciado',
                                    resolve_aestrela(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntLixo, IdasDeposito).
```

Figura 26: A* Seletivo

Os **indicadores de produtividade**, **segunda funcionalidade**, são a quantidade de resíduos recolhidos durante o circuito e a distância média percorrida entre pontos de recolha. Estes indicadores estão presentes em todos os algoritmos e são representados como:

- CL – que representa a quantidade de lixo normal.
- CP - que representa a quantidade de papel e cartão.
- CE - que representa a quantidade de embalagens.
- CV - que representa a quantidade de vidro.
- CO - que representa a quantidade de orgânicos.
- QntTotal – que representa a quantidade total, ou seja, a soma de CL,CP,CE,CV e CO.
- Kms ou Custo – que representam o número médio de quilómetros percorridos.

Através destes indicadores somos capazes de comparar os diferentes circuitos apresentados pelos diversos algoritmos de procura ou simplesmente ficar munidos da informação apresentada pelos indicadores de forma a extrapolar qual foi a produtividade do caminho/circuito apresentado.

As restantes funcionalidades tem todas a mesma essência, isto é, pretendem analisar os circuitos de forma a inferir qual o melhor num determinado aspeto.

A **terceira funcionalidade**, escolher o **circuito mais rápido**, tem por base o critério da distância pois considerou-se correlacionado que o circuito mais rápido está associado ao circuito com a menor distância. Assim sendo, na classe "Funcionalidades.pl" foram implementadas as regras **menorDistanciaDF**, **menorDistanciaDFLimitado**, **menorDistanciaBF**, **menorDistanciaGulosa** e **menorDistanciaAestrela**.

No caso da **pesquisa não informada** utilizou-se o predicado findall de forma a obter todas as soluções alternativas e em seguida a regra auxiliar mínimo, que de uma lista de pares caminho e distância percorrida seleciona o circuito com menores quilómetros percorridos.

Também para a **pesquisa informada** foi passível de implementar esta funcionalidade, uma vez que a heurística utilizada para o algoritmo da procura gulosa e A * é a mesma e tem em conta a distância de cada ponto de recolha ao depósito.

Na seguinte imagem apresentam-se as estratégias de implementação mencionadas.

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

% Teste : menorDistanciaDF(843,921,Resultado).
menorDistanciaDF(I, F,Resultado) :-
                    findall((Solucao,Kms),resolveDFFinal(I, F, Solucao, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                    minimo(L,Resultado).

% Teste : menorDistanciaDFLimitado(355,227,20,Resultado).
menorDistanciaDFLimitado(I,F,Lim,Resultado) :-
                    findall((Solucao,Kms),resolveDFLimite(I, F, Solucao, Lim, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                    minimo(L,Resultado).

% Teste : menorDistanciaBF(355,333,Resultado).
menorDistanciaBF(I,F,Resultado) :-
                    findall((Solucao,Kms),resolveBF(I,F,Solucao, C, Kms,Visitas, QntTotal, IdasDeposito),L),
                    minimo(L,Resultado).

% Teste : menorDistanciaGulosa(355,Caminho/Distancia).
% o destino tem de ser estipulado exemplo goal(965)
menorDistanciaGulosa(Nodo, Caminho/Distancia,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito) :-
                                resolve_gulosa(Nodo, Caminho/Distancia,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).

% Teste : menorDistanciaAestrela(355,Caminho/Distancia).
% o destino tem de ser estipulado exemplo goal(965)
menorDistanciaAestrela(Nodo, Caminho/Distancia, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito) :-
                                resolve_aestrela(Nodo, Caminho/Distancia, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
```

Figura 27: Funcionalidade Circuito Mais Rápido

Por sua vez, a **quarta funcionalidade**, tem o objetivo de encontrar qual o circuito com mais pontos de recolha.

Deste modo pareceu apenas lógico usar estratégias de pesquisa Depth First pois terão maior tendência a apresentar mais nodos devido ao seu algoritmo em profundidade. Todavia, implementou-se também a versão com pesquisa em largura de modo a permitir a comparação de resultados, e deste modo a verificação de que de facto a pesquisa em profundidade encontra mais pontos de recolha por norma. Em adição, visto que as heurísticas dos algoritmos de procura informada lidam com distâncias também não pareceu relevante o uso dos mesmo na resolução da quarta funcionalidade, pelo qual não foram implementadas.

Todas as funcionalidades implementadas, **maisPontosRecolhaDF**, **maisPontosRecolhaDFLimitado** e **maisPontosRecolhaBF**, encontra-se na figura seguinte. Como podemos observar, a implementação passou por através da regra findall retornar uma lista de pares circuito e número de pontos de recolha visitados. De seguida, esta lista passará por um processo de filtragem na regra auxiliar **maisPR** de forma a indicar qual o percurso com mais pontos de recolha visitados.

```
%  Circuitos com mais pontos de recolha

%  Teste : maisPontoRecolhaDF(843, 921,Resultado).
maisPontoRecolhaDF(I, F,Resultado) :- findall((Solucao,Visitas),
                                      resolveDFFinal(I, F, Solucao, C, Kms,Visitas,QntTotal, IdasDeposito),
                                      L),
                                      maisPR(L,Resultado).

% Teste : maisPontoRecolhaDFLimitado(651,662,10,Resultado). Testar com limite 4,6,8 por exemplo
maisPontoRecolhaDFLimitado(I,F,Lim,Resultado) :- findall((Solucao,Visitas),
                                                 resolveDFLimite(I, F, Solucao, Lim, C, Kms,Visitas,QntTotal, IdasDeposito),
                                                 L),
                                                 maisPR(L,Resultado).

% Teste : maisPontoRecolhaBF(651,662,Resultado).
maisPontoRecolhaBF(I,F,Resultado) :- findall((Solucao,Visitas),resolveBF(I,F,Solucao, C, Kms,Visitas, QntTotal, IdasDeposito),L),
                                      maisPR(L,Resultado).
```

Figura 28: Funcionalidade Mais Pontos de Recolha

Visto que o cálculo do findall para a função apenas em profundidade pode ser pesado para pontos distantes entre si, aconselha-se executar os testes pretendidos com recurso a regra em profundidade limitada de modo a obter resultados mais rapidamente. Esta dica funcionará apenas em casos onde a profundidade não seja muito elevada senão o funcionamento será análogo a da função apenas em profundidade, ou seja, a espera por resultados em Swi-Prolog poderá ser longa.

Para finalizar passaremos agora para a **última funcionalidade**, escolher o **circuito mais eficiente**. Para isso foi, antes de mais, necessário definir quais os critérios adequados para considerar um circuito "o mais eficiente". Após alguma reflexão deduziram-se três fatores de análise de eficiência : **quantidade de lixo recolhido**, **quantidade de lixo recolhido por número de idas/descargas ao depósito** e **quantidade de lixo recolhido por quilómetros percorridos**.

Cada um dos critérios apresentados pretende atingir um objetivo e tem por base o seu próprio fator de eficiência. Deste modo, foi preciso pensar que tipos de pesquisa melhor se adequavam as funcionalidades ligadas ao "circuito mais eficiente".

A primeira limitação surgiu com os algoritmos de pesquisa informada, uma vez que já utilizam a heurística da distância. Não fará muito sentido usar estes algoritmos dado que o

objetivo será, dentro da gama de soluções apresentadas pelos mesmos, deduzir a melhor solução de acordo com o fator de eficiência selecionado. Desta forma, considerando que a solução dos algoritmos de pesquisa informada não são "puros", no sentido em que já eles próprios surgem de uma filtragem ou processo heurístico, não se poderia trabalhar com eles de forma a obter somente o critério de eficiência que procuramos usar.

A segunda e última limitação dá-se nos algoritmos de largura pois, como não apresentam uma gama de soluções mas sim apenas uma única, não é possível fazer comparações entre várias possibilidades alternativas e obteríamos sempre as mesmas respostas independentemente do critério de eficiência a testar.

Sobram assim os algoritmos de procura em profundidade, ideais para a aplicação da funcionalidade de escolher o percurso mais eficiente, visto que apresentam uma gama de soluções extensa, a qual permite uma analise de eficiência entre percursos alternativos. Visto a abrangência da obtenção de soluções alternativas aconselha-se o uso do **algoritmo em profundidade limitado**, pois serve perfeitamente para analisar os resultados dos vários critérios de eficiência implementados e obtém rapidamente, desde que a profundidade se mantenha pequena, o "caminho mais eficiente".

Comecemos a demonstração do código com o critério de eficiência : "quantidade de lixo recolhido", ou seja, o objetivo aqui será simplesmente saber qual o percurso com mais lixo recolhido. O critério de eficiência prende-se então apenas a ver que percurso consegue recolher mais lixo entre o ponto inicial e o ponto de destino.

Assim sendo, na figura 29 surgem as regras **maisLixoDF** e **maisLixoDFLimitado**, onde se codificaram o raciocínio anteriormente referido. Podemos ver que ambas recorrem ao predicado findall e a respetiva função algorítmica em profundidade para apresentar uma lista de caminho alternativos, onde posteriormente com auxilio da regra auxiliar **maisQnt** será filtrada o melhor circuito.

```
% Eficiencia Por Quantidade de Lixo Recolhido

% Teste :  maisLixoDF(843, 921, Resultado).
maisLixoDF(I, F, Resultado) :- findall((Solucao,QntTotal,IdasDeposito),
                    resolveDFFinal(I, F, Solucao, C, Kms,Visitas,QntTotal, IdasDeposito),
                    L),
                    maisQnt(L,Resultado).

% Teste : maisLixoDFLimitado(355,921,10,Resultado).
maisLixoDFLimitado(I,F,Lim,Resultado) :- findall((Solucao,QntTotal,IdasDeposito),
                    resolveDFLimite(I, F, Solucao, Lim, C, Kms,Visitas,QntTotal, IdasDeposito),
                    L),
                    maisQnt(L,Resultado).
```

Figura 29: Critério Eficiência - Maior Quantidade Lixo Recolhido

De seguida, passo a demonstração do código com o critério de eficiência : "quantidade de lixo recolhido por número de idas/descargas ao depósito". Neste caso, o objetivo por detrás do critério de eficiência **é mais complexo**. Ao considerarmos o número de idas ao depósito estamos a avaliar não só o número de desvios efetuados mas também qual a percentagem de eficiência relativamente ao uso da carga máxima disponível pelo camião de recolha de resíduos urbanos. Desta forma, podemos estabelecer um paralelismo entre o número médio de carga utilizada durante o circuito (quantidade de lixo coletada / número de idas ao depósito) e a carga

máxima permitida pelo camião (15000 Litros). E através da razão entre eles, (carga média circuito/ carga total camião), saber qual a percentagem de eficiência do uso da carga do camião de recolha.

Assim sendo, na figura 30 surgem as regras **maisLixoIdasDepositoDF** e **maisLixoIdasDepositoDFLimitado**, onde se codificaram o raciocínio anteriormente referido. Podemos ver que ambas recorrem ao predicado findall e a respetiva função algorítmica em profundidade para apresentar uma lista de caminho alternativos, onde posteriormente com auxilio da regra auxiliar **maisQntE** será filtrada o melhor circuito com base no critérios já mencionados.

```
% Eficiencia Por Quantidade de Lixo e Idas ao Deposito

% Teste :  maisLixoIdasDepositoDF(843, 921, Resultado).
maisLixoIdasDepositoDF(I, F, Resultado) :- findall((Solucao,QntTotal,IdasDeposito),
                        resolveDFFinal(I, F, Solucao, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                        maisQntE(L,Resultado).

% Teste : maisLixoIdasDepositoDFLimitado(355,921,10,Resultado).
maisLixoIdasDepositoDFLimitado(I,F,Lim,Resultado) :- findall((Solucao,QntTotal,IdasDeposito),
                        resolveDFLimite(I, F, Solucao, Lim, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                        maisQntE(L,Resultado).
```

Figura 30: Critério Eficiência - Quantidade de Lixo Recolhido por Idas ao Depósito

Finalmente surge a demonstração do código com o critério de eficiência : "quantidade de lixo recolhido por quilómetros percorridos", ou seja, o objetivo passa por saber qual o percurso com mais lixo recolhido em menos quilómetros. Este critério de eficiência pretende descobrir qual o percurso com menor distância inicial entre o ponto de destino que consegue recolher mais lixo.

Assim sendo, na figura 31 surgem as regras **maisLixoKmsDF** e **maisLixoKmsDFLimitado**, onde se codificou o raciocínio anteriormente referido. Podemos ver que ambas recorrem ao predicado findall e a respetiva função algorítmica em profundidade para apresentar uma lista de caminho alternativos, onde posteriormente com auxilio da regra auxiliar **maisQntKm** e de acordo com as especificidades mencionadas será filtrada o melhor circuito.

```
% Eficiencia Por Quantidade de Lixo Recolhido e Kms percorridos

% Teste :  maisLixoKmsDF(843, 921, Resultado).
maisLixoKmsDF(I, F, Resultado) :- findall((Solucao,QntTotal,Kms),
                        resolveDFFinal(I, F, Solucao, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                        maisQntKm(L,Resultado).

% Teste : maisLixoKmsDFLimitado(355,921,10,Resultado).
maisLixoKmsDFLimitado(I,F,Lim,Resultado) :- findall((Solucao,QntTotal,Kms),
                        resolveDFLimite(I, F, Solucao, Lim, C, Kms,Visitas,QntTotal, IdasDeposito),L),
                        maisQntKm(L,Resultado).
```

Figura 31: Critério Eficiência - Quantidade de Lixo Recolhido por Kms Percorridos

Dá-se então por concluída a apresentação das funcionalidades. De seguida, passamos para o capitulo das regras auxiliares.

REGRAS AUXILIARES

Com o propósito de agrupar as diversas regras que serviram de auxílio às funcionalidades e algoritmos do sistema foi criado o ficheiro PROLOG "RegrasAuxiliares.pl". Este ficheiro divide-se em três categorias principais : **regras auxiliares**, **Regras Auxiliares QUERIES** e **Regras Auxiliares Algoritmos**.

Nas **regras auxiliares** temos funções auxiliares mais genéricas e diretas como membro, verificaElem, soluções, comprimento, não, inverso e seleciona.

```
%--------------------------------- - - - - - - - - - -
%--------------------------------- - - - - - - - - - -
%- REGRAS AUXILIARES

%--------------------------------- - - - - - - - - - -
% Extensao do meta-predicado membro: Elemento, Lista -> {

membro(X,LD) :- member(X,LD).
%--------------------------------- - - - - - - - - - -
% Verifica se existe pelo menos um elemento de uma lista

verificaElem([X|H],LD) :- membro(X,LD).
verificaElem([X|H],LD) :- verificaElem(H,LD).
%--------------------------------- - - - - - - - - - -
% Extensao do meta-predicado solucoes: Elemento, Questao,

solucoes(F,P,S) :- findall(F,P,S).
%--------------------------------- - - - - - - - - - -
% Extensao do meta-predicado comprimento: Lista, Comprime

comprimento(L,N):- length(L,N).
```

Figura 33: Regras Auxiliares 1

```
%--------------------------------- - - - - - - - - - -
% Extensao do meta-predicado nao: Questao -> {V,F}

nao( Questao ) :-
    Questao, !, fail.
nao( Questao ).
%--------------------------------- - - - - - - - - - -
% Calcula o inverso de uma lista: Lista -> {V,F}

inverso(Xs, Ys) :-
    inverso(Xs, [], Ys).

inverso([], Xs, Xs).
inverso([X|Xs], Ys, Zs) :- inverso(Xs, [X|Ys], Zs).
%--------------------------------- - - - - - - - - - -
% Seleciona um elemento de uma lista  -> {V,F}

seleciona(E,[E|Xs],Xs).
seleciona(E,[X|Xs],[X|Ys]) :- seleciona(E,Xs,Ys).
```

Figura 32: Regras Auxiliares

Nas **regras auxiliares QUERIES** temos funções auxiliares mais especificas, neste caso orientadas as funcionalidades. Em concreto temos cinco regras auxiliares : maisPR, mínimo e maisQnt, maisQntE e maisQntKm. A regra maisPR serve para dado uma lista de caminhos selecionar aquele com mais pontos de recolha.

```
% Regras Auxiliares QUERIES
%--------------------------------- - - - - - - - - - -  -  -  -  -
% Predicado que calcula o caminho com mais pontos de recolha
% maior : Lista , Elemento -> {V,F}

maisPR([E],E).

maisPR([(S,Nr)|H],(S,Nr)) :-
    maisPR(H,(S1,Nr1)),
    Nr > Nr1.

maisPR([(S,Nr)|H],(S1,Nr1)) :-
    maisPR(H,(S1,Nr1)),
    Nr1 >= Nr.
```

Figura 34: Regras auxiliares queries - mais pontos recolha

A regra mínimo serve para dado uma lista de caminhos selecionar o circuito mais rápido, ou seja, com base no critério de distância escolher o que o percurso tem menos quilómetros.

```
%-------------------------------- - - - - - - - - - - -  -  -  -  -  -
% Predicado que cálcula o elemento minimo de uma lista
% minimo : Lista , Elemento -> {V,F}
minimo([(P,X)],(P,X)).
minimo([(Px,X)|L], (Py,Y)) :- minimo(L,(Py,Y)), X > Y.
minimo([(Px,X)|L], (Px,X)) :- minimo(L,(Py,Y)), X =< Y.
```

Figura 35: Regras Auxiliares queries - circuito mais rápido

A regra maisQnt serve para escolher o circuito mais eficiente (usando o critério quantidade lixo recolhido e número de idas ao deposito). Na realidade a função apenas tem em conta o número de litros de lixo recolhido mas pode ser efetuada um análise desse valor juntamente com o número de visitas ao depósito, considerando-se assim mais eficiente não o camião que recolhe mais lixo mas sim na realidade o camião que aproveita melhor a capacidade estabelecida pelo limite máximo de lixo coletado. Uma métrica alternativa poderia ser o camião que recolhe mais lixo por quilómetros percorridos.

```
%-------------------------------- - - - - - - - - - -  -  -
% Predicado que calcula o caminho com mais lixo recolhido
% maior : Lista , Elemento -> {V,F}

maisQnt([E],E).

maisQnt([(S,Qnt,Nr)|H],(S,Qnt,Nr)) :-
    maisQnt(H,(S1,Qnt1,Nr1)),
    Qnt > Qnt1.

maisQnt([(S,Qnt,Nr)|H],(S1,Qnt1,Nr1)) :-
    maisQnt(H,(S1,Qnt1,Nr1)),
    Qnt1 >= Qnt.
```

Figura 36: Regras Auxiliares queries - mais lixo recolhido

```
%-------------------------------- - - - - - - - - -  -  -  -  -  -
% Predicado que calcula o caminho com mais lixo recolhido por idas ao deposito
% maisQntE : Lista , Elemento -> {V,F}

maisQntE([E],E).

maisQntE([(S,Qnt,Nr)|H],(S,Qnt,Nr)) :-
    maisQntE(H,(S1,Qnt1,Nr1)),
    (Qnt/Nr) > (Qnt1/Nr1).

maisQntE([(S,Qnt,Nr)|H],(S1,Qnt1,Nr1)) :-
    maisQntE(H,(S1,Qnt1,Nr1)),
    (Qnt1/Nr1) >= (Qnt/Nr).
```

Figura 38: Regras Auxiliares queries - Mais quantidade lixo por idas ao deposito

```
% Predicado que calcula o caminho com mais lixo recolhido por Kms percorridos
% maisQntKm : Lista , Elemento -> {V,F}

maisQntKm([E],E).
maisQntKm([(S,Qnt,Km)|H],(S,Qnt,Km)) :- maisQntKm(H,(S1,Qnt1,Km1)),
                                        (Qnt/Km) > (Qnt1/Km1).
maisQntKm([(S,Qnt,Km)|H],(S1,Qnt1,Km1)) :- maisQntKm(H,(S1,Qnt1,Km1)),
                                          (Qnt1/Km1) >= (Qnt/Km).
```

Figura 37: Regras Auxiliares queries - Mais quantidade lixo por kms percorridos

Por último nas **regras auxiliares Algoritmos** temos regras que sustentam o correto funcionamento dos algoritmos. Comecemos pelo predicado **goal** que define o depósito ou destino final do algoritmo, na figura 39 indica-se o goal(965) que como exemplo demonstrativo serviu para definir o deposito, que será usado tanto em algoritmos de procura como em outras regras apresentadas a seguir.

```
%--------------------------------
%--------------------------------
% % Regras Auxiliares Algoritmos
%--------------------------------
% deposito
goal(965).
%--------------------------------
```

Figura 39: Predicado goal

Outras regras de extrema relevância são as regras que definem como funciona a adjacência dos algoritmos, podendo esta ser unidirecional para uns algoritmos e bidirecional para outros como já foi apresentado anteriormente, desta forma nasceram as regras **adjacente** e **adjacente2**, respetivamente. A regra **adjacente3** também tem o mesmo princípio, mas visto usar internamente a regra adjacente2 e usar métricas como custos a sua implementação manteve-se na classe "algoritmos.pl" onde já foi estudada em detalhe.

```
% predicado adjacente para grafos unidirecional

adjacente(Nodo,ProxNodo) :-
    ponto(_,_,Nodo,_,Vizinhos,_,_,_,_,_),
    membro(ProxNodo,Vizinhos).
% adjacente(Nodo,ProxNodo) :-
%    ponto(_,_,ProxNodo,_,Vizinhos,_,_,_,_,_),
%    membro(Nodo,Vizinhos).

% predicado adjacente para grafos bidirecional

adjacente2(Nodo,ProxNodo) :-
    ponto(_,_,Nodo,_,Vizinhos,_,_,_,_,_),
    membro(ProxNodo,Vizinhos).
adjacente2(Nodo,ProxNodo) :-
    ponto(_,_,ProxNodo,_,Vizinhos,_,_,_,_,_),
    membro(Nodo,Vizinhos).
```

Figura 40: Regras adjacente e adjacente2

Um dos tópicos mais abordados nas regras auxiliares para algoritmos prende-se aos cálculos de distância, ora não fosse a heurística dos algoritmos de pesquisa informada baseada em distância, uma das metas de produtividade o número de quilómetros percorridos e existisse um fator de carga máxima do camião que obrigasse a desvios ao deposito que dependam

intrinsecamente do cálculo das distâncias. Assim, com o auxílio das regras : **kilometros**, **estima** e **distancia**, foi possível cumprir com os objetivos propostos. A regra **kilometros** é bastante simples, dado um percurso calcula a distância entre dois pontos sucessivos desse percurso através da função distancia. A regra **distancia** recebe como argumento dois nodos, que com base nas suas coordenadas geográficas e de cálculos matemáticos apresenta uma boa aproximação do valor da distância entre pontos em quilómetros. Finalmente, na figura 41 conseguimos ainda observar a implementação da regra **estima**, que é em todo o processo de cálculo semelhante a regra distancia, a diferença é que o calculo será sempre a distância de um nodo ao depósito e desta forma apenas precisa de saber qual o nodo em questão, pois usa internamente o predicado goal para aceder as coordenadas do ponto depósito.

```
% calcula kilometros de um percurso
kilometros([H],0).
kilometros([H,B|T],Kms) :- distancia(H, B, Dist),
                           kilometros([B|T],KmAux),
                           Kms is KmAux+Dist.

% estima distancia entre nodo e goal/deposito definido
estima(Nodo, Dist) :-
    goal(D),
    ponto(Lat2,Lon2,D,_,_,_,_,_,_,_),
    ponto(Lat1,Lon1,Nodo,_,_,_,_,_,_,_),
    P is 0.017453292519943295,
    A is (0.5 - cos((Lat2 - Lat1) * P) / 2 + cos(Lat1 * P) * cos(Lat2 * P) * (1 - cos((Lon2 - Lon1) * P)) / 2),
    Dist is (12742 * asin(sqrt(A))).

% estima distancia entre dois nodos
distancia(Nodo, NodoT, Dist) :-
    ponto(Lat2,Lon2,Nodo,_,_,_,_,_,_,_),
    ponto(Lat1,Lon1,NodoT,_,_,_,_,_,_,_),
    P is 0.017453292519943295,
    A is (0.5 - cos((Lat2 - Lat1) * P) / 2 + cos(Lat1 * P) * cos(Lat2 * P) * (1 - cos((Lon2 - Lon1) * P)) / 2),
    Dist is (12742 * asin(sqrt(A))).
```

Figura 41: Regras auxiliares para calculo de distancias

Finalmente mas não menos importante, surgem as regras **qntLixos** e **deposito**. Estas regras auxiliares, evidentes nas figuras 42 e 43, fornecem métricas bastante enriquecedores sobre o percurso efetuado.

Em relação à **qntLixos** o seu funcionamento permite saber para cada viagem a quantidade de lixo recolhido de forma seletiva, ou seja, diferencia as quantidades coletadas de lixo normal, papel e cartão, embalagens, vidros e matéria orgânica.

```
% calcula quantidade de lixo de um percurso
qntLixos([],0,0,0,0,0).
qntLixos([H|T],CL,CP,CE,CV,CO) :- ponto(_,_,H,_,_,CLP,CPP,CEP,CVP,COP),
                                  qntLixos(T,CLAux,CPAux,CEAux,CVAux,COAux),
                                  CL is CLAux+CLP,    % Quantidade lixo tipo - geral
                                  CP is CPAux+CPP,    % Quantidade lixo tipo - papel e cartao
                                  CE is CEAux+CEP,    % Quantidade lixo tipo - embalagens
                                  CV is CVAux+CVP,    % Quantidade lixo tipo - vidro
                                  CO is COAux+COP.    % Quantidade lixo tipo - organio
```

Figura 42:Regra qntLixos

Já noutra vertente, a regra **deposito** serve o propósito de indicar o número de idas ao depósito de um determinado percurso, este valor é usado como um critério de eficiência do camião de recolha pois se dividir-mos a quantidade de lixo recolhido pelo número de idas ao depósito temos um **indicador da carga média recolhida** nas viagens, que comparando com a carga máxima permitida pelo camião (15000 Litros) conseguimos inferir a **percentagem média de ocupação** da carga do camião, por exemplo podemos considerar que uma ocupação média superior a 90% seja uma boa eficiência em termos de uso da carga máxima, isto pois quanto melhor utilizarmos a carga máxima disponível menos desvios ao depósito serão necessários.

```
% apresenta solucao com numero de idas ao deposito
deposito(Percurso,Solucao, DepCamiao,Idas) :-
    goal(D),
    depositoAux(Percurso,[D],Solucao,DepCamiao,Idas).

depositoAux([],Solucao,Solucao,0,1).
depositoAux([H|Tail],Aux,Solucao,Carga,Idas) :-
    goal(D),
    ponto(_,_,H,_,_,Q1,Q2,Q3,Q4,Q5),
    depositoAux(Tail,Lista,Solucao,CargaAux,IdasAux),
    ( CargaAux+Q1+Q2+Q3+Q4+Q5 =< 14000 -> Lista = [H|Aux],
      Carga is CargaAux+Q1+Q2+Q3+Q4+Q5,
      Idas is IdasAux;
      Lista = [D,H|Aux], Carga is 0 , Idas is IdasAux + 1).
```

Figura 43: Regra deposito e regra depositoAux

ANÁLISE DE RESULTADOS

Nesta secção serão apresentados os resultados das diversas estratégias de procura implementadas no projeto, como forma de demonstrar a consistência entre os resultados obtidos e o pretendido. Cada algoritmo apresentado foi aplicado à base de conhecimento presente neste relatório que, como já fora mencionado, corresponde ao grafo pelo qual os algoritmos iteram.

RESULTADOS – TESTES

Com o intuito de apresentar uma análise comparativa dos resultados obtidos dos diferentes algoritmos de pesquisa e de demonstrar qual a melhor estratégia para cada funcionalidade implementada, seguem-se alguns cenários de teste em SWI-Prolog, exemplificativos de como funciona o programa e representativos de todo o trabalho até agora descrito.

ALGORITMOS

Foi utilizado o mesmo teste em todos os algoritmos para garantir uma homogeneidade entre as soluções obtidas e por sua vez garantir que possam ser passíveis de comparação. Nos demais testes podemos ver indicadores de produtividade e a realização completa do caso proposto, visto ter em conta desvios ao depósito quando o camião de recolha de resíduos urbanos atinge a carga máxima.

DFS

Comecemos pelos algoritmos de pesquisa não informada, de seguida conseguimos ver o resultado obtido a partir do algoritmo DepthFisrt, onde aparecem : todos os IDS dos pontos de recolha visitados durante o percurso (incluindo o inicial e o final), o número de quilómetros percorridos, quantas idas foram feitas ao depósitos e as quantidades recolhidas de cada tipo de lixo. Na imagem confirma-se que o primeiro resultado obtido é o de menor custo ou distância mas isto não é uma garantia do algoritmo. Podemos ainda observar o resultado de uma solução alternativa mas que devido a ter um circuito maior obtêm mais lixo recolhido e por sua vez ocorreu a necessidade de ir ao depósito descarregar a carga (podemos ver que a quantidade total de lixo recolhido foram 23750 litros > que 15000 litros) e que o número de quilómetros percorridos foi também consideravelmente superior.

```
?- resolveDFFinal(333, 921, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1 ;
Solucao = [333,334,345,346,642,531,644,467,494,233,265,231,66,214,180,225,245,267,163,227,4
8,965,121,205,85,290,692,694,938,944,843,921,965],
CL = 10600,
CP = 2350,
CE = 2930,
CV = 6590,
CO = 1280,
Kms = 15.226022893957948,
Visitas = 33,
QntTotal = 23750,
IdasDeposito = 2 .
```

Figura 44: Teste Swi-Prolog DF

DFS-Limite

Na próxima imagem segue-se o resultado obtido a partir do algoritmo em Profundidade mas Limitado, tudo funciona de maneira semelhante ao caso anterior só que podemos reparar que existe também um limite definido a titulo demonstrativo como 33. As soluções apresentadas são, tal como previsto, iguais as do caso anterior.

```
?- resolveDFLimite(333,921,Solucao,33, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito).

Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1 ;
Solucao = [333,334,345,346,642,531,644,467,494,233,265,231,66,214,180,225,245,267,163,227,4
8,965,121,205,85,290,692,694,938,944,843,921,965],
CL = 10600,
CP = 2350,
CE = 2930,
CV = 6590,
CO = 1280,
Kms = 15.226022893957948,
Visitas = 33,
QntTotal = 23750,
IdasDeposito = 2 .
```

Figura 45: Teste Swi-Prolog DF Limitado

Contudo se limitarmos a profundidade máxima a 21, o resultado da segundo caminho apresentado já mudará uma vez que para isso seria necessário conseguir atingir uma profundidade de 33, ou seja, o numero de pontos do circuito anterior. Na próxima figura comprova-se esta afirmação.

```
?- resolveDFLimite(333,921,Solucao,21, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito).

Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1 ;
Solucao = [333,334,345,346,642,531,644,467,469,474,843,921,965],
CL = 6480,
CP = 1480,
CE = 690,
CV = 440,
CO = 90,
Kms = 2.508233571522562,
Visitas = 13,
QntTotal = 9180,
IdasDeposito = 1 .
```

Figura 46: Teste Swi-Prolog Profundidade Limitada 21

BFS

Ainda no tópico de pesquisa não informada, consegue-se observar na figura 47 que o resultado da pesquisa em largura é também o ótimo esperado. Nela figuram os mesmos campos apresentados nos algoritmos supramencionados.

```
?- resolveBF(333,921,Caminho, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
Caminho = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1 .
```

Figura 47: Teste Swi-Prolog BF

Gulosa

Quanto a pesquisa informada, comecemos pela pesquisa gulosa, figura 48, podemos ver os mesmos campos presentes nos algoritmos anteriores com a adição do campo custo, que é nada mais nada menos que o custo calculado pela heurística. O resultado retornado pela regra não é o circuito ótimo.

Como observação final, podemos notar que o ID 965 repete-se duas vezes no final do circuito apresentado. Isto ocorre pois 965 é tanto o ponto destino do algoritmo como o ponto do depósito, logo no final de ir ao destino 965 o algoritmo apresenta sempre o depósito que mais uma vez será 965.

```
% c:/Users/LuisPinto/Desktop/SRCR/TP_INDV/Funcionalidades.pl compiled 0.00 sec, 174 clauses
?- resolve_gulosa(333, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito).
Solucao = [333,662,351,346,345,921,965,965],
Custo = 1.3671224305214018,
CL = 2350,
CP = 470,
CE = CO, CO = 90,
CV = 140,
Kms = 1.3671224305214016,
Visitas = 8,
QntTotal = 3140,
IdasDeposito = 1 
```

Figura 48: Teste Swi-Prolog Gulosa

A*

Por fim, no último caso de algoritmos de pesquisa heurística temos o resultado do algoritmo A*. De facto apresenta a melhor solução em termos de custo/distância, tal como podemos ver depois de apresentar a melhor solução continuará a apresentar ordenadamente as restantes soluções, da segunda melhor até a pior encontrada. Se compararmos o resultado obtido pelo algoritmo A* com o resultado do algoritmo da pesquisa Gulosa torna-se evidente que a pesquisa gulosa não apresenta a melhor solução. Esta é então referida em A* com um custo de 1.19 Kms e perfaz um total de 9050 litros de lixo recolhido, não necessitando assim de desvios intermédios ao depósito.

```
?- resolve_aestrela(333, Caminho/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito
).
Caminho = [333,386,651,649,647,644,843,921,965,965],
Custo = 1.1958694602443538,
CL = 6240,
CP = 1590,
CE = 690,
CV = 440,
CO = 90,
Kms = 1.195869460244354,
Visitas = 10,
QntTotal = 9050,
IdasDeposito = 1 ;
Caminho = [333,386,651,649,647,644,469,843,921,965,965],
Custo = 1.1968666259682688,
CL = 7370,
CP = 1960,
CE = 690,
CV = 440,
CO = 90,
Kms = 1.196866625968269,
Visitas = 11,
QntTotal = 10550,
IdasDeposito = 1 ,
```

Figura 49: Teste Swi-Prolog A Estrela

Comparando as estratégias existe pois um justo vencedor no que a apresentação do percurso com menor custo, o algoritmo de pesquisa informada A*.

Medição de Performance dos Algoritmos

De modo a preencher a tabela 3, de analise comparativa de estratégias de procura, foi necessário recorrer a algumas ferramentas disponíveis no PROLOG, em particular as regras **time** e **statistics**. De seguidas são apresentadas as regras utilizadas presentes no ficheiro "RegrasAuxiliares.pl".

```
%--------------------------------
% % Regras Auxiliares Algoritmos  - MEDICAO PERFORMANCE
%--------------------------------
% Depth First - Estatisticas de Memoria utilizada e Tempo de execucao.
statisticsDF(I, F, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito,Mem) :-
    statistics(global_stack, [Used1,Free1]),
    time(resolveDFFinal(I, F, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito)),
    statistics(global_stack, [Used2,Free2]),
    Mem is Used2 - Used1.
%--------------------------------
% % Depth First Limitado- Estatisticas de Memoria utilizada e Tempo de execucao.
statisticsDFLimitado(I,F,Solucao,Lim, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito,Mem) :-
    statistics(global_stack, [Used1,Free1]),
    time(resolveDFLimite(I,F,Solucao,Lim, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDeposito)),
    statistics(global_stack, [Used2,Free2]),
    Mem is Used2 - Used1.
%--------------------------------
% Breadth First - Estatisticas de Memoria utilizada e Tempo de execucao.
statisticsBF(I,F,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito,Mem) :-
    statistics(global_stack, [Used1,Free1]),
    time(resolveBF(I,F,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito)),
    statistics(global_stack, [Used2,Free2]),
    Mem is Used2 - Used1.
%--------------------------------
% Gulosa - Estatisticas de Memoria utilizada e Tempo de execucao.
statisticsGulosa(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito,Mem) :-
    statistics(global_stack, [Used1,Free1]),
    time(resolve_gulosa(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito)),
    statistics(global_stack, [Used2,Free2]),
    Mem is Used2 - Used1.
%--------------------------------
% A estrela- Estatisticas de Memoria utilizada e Tempo de execucao.
statisticsAEstrela(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito, Mem) :-
    statistics(global_stack, [Used1,Free1]),
    time(resolve_aestrela(Nodo, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito)),
    statistics(global_stack, [Used2,Free2]),
    Mem is Used2 - Used1.
```

Figura 50: Regras Auxiliares para Estatísticas de memoria e tempo

A aplicação destas regras traduziu-se na realização dos mesmos testes executados anteriormente, agora tendo em consideração fatores como utilização de memória e tempo de execução. Assim, chegaram-se aos seguintes resultados presentes nas figuras 51 a 55 (ter apenas em atenção os campos seconds e Mem):

```
?- statisticsDF(333, 921, Solucao,CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito,Mem).

% 140 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1,
Mem = 8584 ;
% 1,434 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,334,345,346,642,531,644,467,494,233,265,231,66,214,180,225,245,267,163,227,4
8,965,121,205,85,290,692,694,938,944,843,921,965],
CL = 10600,
CP = 2350,
CE = 2930,
CV = 6590,
CO = 1280,
Kms = 15.226022893957948,
Visitas = 33,
QntTotal = 23750,
IdasDeposito = 2,
Mem = 15320 .
```

Figura 51: Estatísticas DF

```
?- statisticsDFLimitado(333,921,Solucao,21, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal,IdasDepos
ito,Mem).
% 143 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1,
Mem = 8592 ;
% 829 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,334,345,346,642,531,644,467,469,474,843,921,965],
CL = 6480,
CP = 1480,
CE = 690,
CV = 440,
CO = 90,
Kms = 2.508233571522562,
Visitas = 13,
QntTotal = 9180,
IdasDeposito = 1,
Mem = 46872 .
```

Figura 52: Estatísticas DF Limitado

```
?- statisticsBF(333,921,Solucao, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito,Mem).
% 3,608 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,334,345,921,965],
CL = 930,
CP = CE, CE = CO, CO = 90,
CV = 140,
Kms = 1.330954464760197,
Visitas = 5,
QntTotal = 1340,
IdasDeposito = 1,
Mem = 18304 .
```

Figura 53: Estatísticas BF

```
?- statisticsGulosa(333, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposito
,Mem).
% 2,285 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
Solucao = [333,662,351,346,345,921,965,965],
Custo = 1.3671224305214018,
CL = 2350,
CP = 470,
CE = CO, CO = 90,
CV = 140,
Kms = 1.3671224305214016,
Visitas = 8,
QntTotal = 3140,
IdasDeposito = 1,
Mem = 26552
```

Figura 54: Estatísticas Gulosa

```
?- statisticsAEstrela(333, Solucao/Custo, CL/CP/CE/CV/CO, Kms,Visitas, QntTotal, IdasDeposi
to, Mem).
% 836,102 inferences, 0.109 CPU in 0.104 seconds (105% CPU, 7644361 Lips)
Solucao = [333,386,651,649,647,644,843,921,965,965],
Custo = 1.1958694602443538,
CL = 6240,
CP = 1590,
CE = 690,
CV = 440,
CO = 90,
Kms = 1.195869460244354,
Visitas = 10,
QntTotal = 9050,
IdasDeposito = 1,
Mem = 5777088 ;
% 105,072 inferences, 0.016 CPU in 0.011 seconds (142% CPU, 6724608 Lips)
Solucao = [333,386,651,649,647,644,469,843,921,965,965],
Custo = 1.1968666259682688,
CL = 7370,
CP = 1960,
CE = 690,
CV = 440,
CO = 90,
Kms = 1.196866625968269,
Visitas = 11,
QntTotal = 10550,
IdasDeposito = 1,
Mem = 8662080 ,
```

Figura 55: Estatísticas A Estrela

ANÁLISE - COMENTÁRIOS FINAIS

Dada por concluída a demonstração da execução dos algoritmos surge agora a necessidade de os categorizar com base em critérios de performance, possibilitando assim a comparação entre as diferentes estratégias de procura. A seguinte tabela apresenta os critérios de comparação escolhidos para medidas de performance, que culminaram em : tempos de execução, utilização de memória, profundidade/custo e se encontra a melhor solução.

Estratégia	Tempo (Segundos)	Espaço	Profundidade\Custo (Kms)	Encontrou melhor solução?
DFS	0.000	8584	1.331	Não
DFS-Limite	0.000	8592	1.331	Não
BFS	0.000	18304	1.331	Não
Gulosa	0.000	26552	1.367	Não
A*	0.104	5777088	1.196	Sim

Tabela 3 : Análise Comparativa Estratégias de Procura

Relativamente a tabela 3, podemos concluir que o melhor algoritmo é o A* pois de facto é o único que atinge a solução ótima. Isto reflete-se em termos do espaço utilizado pelo algoritmo que é relativamente superior quando comparado com os restantes. Desta forma, também é o que apresenta o menor custo, ou seja, a menor distância. O que faz sentido, tendo em conta que a heurística implementada tem em conta a distância em cada passo calculado. Este algoritmo é também reflete o seu bom funcionamento no tempo de execução demorando mais que os restantes algoritmos. Em adição, tal como era esperado, nota-se que os algoritmos em profundidade são os que ocupam menos espaço. Precisamos também de avaliar estes resultados com um grão de sal e entender que o algoritmo A* funciona a pensar nas ligações futuras e está um passo a frente dos algoritmos não informados, desta forma é natural que o resultado apresentado pelos algoritmos em largura e em profundidade não tenham conseguido atingir a solução ótima encontrada pelo algoritmo A*.

CONCLUSÃO E SUGESTÕES

Dado por concluído o projeto individual, considero relevante efetuar uma análise crítica do trabalho realizado, apontando os aspetos positivos, as dificuldades sentidas durante a sua realização e como foram superadas, e por fim perspetivas para um trabalho futuro.

No que diz respeito aos aspetos positivos, o facto de o trabalho realizado ser completo, no sentido em que cumpre com todas as funcionalidades estipuladas no relatório. Além disso, a opção de realizar o trabalho na integra invés da versão reduzida foi um aspeto desafiante mas muito realizador. Relativamente a documentação do relatório destaca-se a sua composição e robustez, permitindo a sua arquitetura uma representação clara de todas as etapas do projeto.

Por outro lado, relativamente às dificuldades sentidas, o tratamento de dados do dataset revelou ser o maior obstáculo devido a inerente complexidade da sua análise. Porém, depois de entendidos os seus campos relevantes, o tratamento do mesmo não passou de um mero programa em java capaz de originar uma base de conhecimento.

Numa fase futura, tenciono potencializar o projeto implementando mais algoritmos de pesquisa, quer não informada quer informada, visando com estes métodos o enriquecimento do produto final.

Constato que o resultado final se revela deleitoso visto que o programa é capaz de implementar os diversos algoritmos de procura/pesquisa, ter em conta os fatores de produtividade - a quantidade recolhida e a distância média percorrida entre pontos de recolha, gerar circuitos de recolha tanto indiferenciada como seletiva e ainda proporcionar várias funcionalidades relativas aos circuitos (tais como a identificação do circuito com mais pontos de recolha, mais rápido ou mais eficiente).

Para finalizar, concluo que o balanço do caso prático foi bastante positivo uma vez que as dificuldades sentidas foram ultrapassadas e o programa apresentado está em concordância com as orientações fornecidas e objetivos propostos.

REFERÊNCIAS

REFERÊNCIAS BIBLIOGRÁFICAS

[Analide, 2011] ANALIDE, Cesar, Novais, Paulo, Neves, José,
"Sugestões para a Elaboração de Relatórios",
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

[Russel,Norvig, 2009] RUSSEL, Stuart, NORVIG, Peter
"Artificial Intelligence - A Modern Approach, 3rd edition",
Pearson, 2009.

[Costa, Simões , 2008] COSTA, Ernesto, SIMÕES, Anabela,
"Inteligência Artificial-Fundamentos e Aplicações",
FCA, 2008.

[Lazebnik, 2017] LAZEBNIK, Svetlana,
"Lecture notes Fall 2017 Artificial Intelligence",
University of Illinois, 2017.

[Reis , 2019] REIS, Luís Paulo,
"Lecture notes Artificial Intelligence",
Universidade do Porto, 2019.

SIGLAS E ACRÓNIMOS

BC	Base de Conhecimento
DF	Depth First
BF	Breadth First
UC	Unidade Curricular
SRCR	Sistemas de Representação de Conhecimento e Raciocínio