Universidade do Minho
Escola de Engenharia

Universidade do Minho

Departamento de Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Prático Individual

Relatório de Desenvolvimento

Maria Ramos
a89541

9 de setembro de 2021

Conteúdo

Lista de Figuras

Listings

Capítulo 1

Introdução

Este trabalho foi realizado no âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio do 3º ano do 2º semestre do curso de Engenharia Informática.

Foi proposto o desenvolvimento de um sistema de recomendação de percursos para recolha de resíduos.

O trabalho a seguir apresentado tem em consideração a capacidade máxima dos transportes.

Capítulo 2

Desenvolvimento

2.1 Formulação do Problema

O sistema a desenvolver tem o intuito de recomendar circuitos de recolha de resíduos urbanos no concelho de Lisboa. A resolução deste problema consiste na resolução de um **problema de pesquisa**, já que é necessário encontrar a sequência de ações que leva a um estado desejável, isto é, as ações que levarão o transporte de resíduos de um ponto inicial (garagem) a um ponto final (depósito).

Um problema de pesquisa pode ser definido formalmente em cinco componentes:

- Estado Inicial
- Estado Objetivo
- Representação do Estado
- Operadores
- Custo da solução

No contexto deste problema temos, então:

- **Estado Inicial**: Garagem
- **Estado Objetivo**: Depósito
- **Estados**: Os vários pontos de recolha
- **Operações**: Conduzir entre os pontos de recolha; Recolher resíduos
- **Solução**: Uma sequência de pontos de recolha
- **Custo da solução**: Será calculado através de diferentes medidas de desempenho como distância média entre pontos de recolha, número de pontos de recolha ou quantidade de lixo recolhida.

Sabe-se que após o transporte despejar os resíduos no depósito, deverá voltar à garagem, no entanto, como nesse percurso não será feita recolha de resíduos, este não foi considerado. O impacto que o retorno à garagem teria no custo da solução não tem grande interesse na análise dos percursos com base nos fatores de produtividade fornecidos.

Este problema define-se como um problema de estado único, pois o ambiente é determinístico e totalmente observável, isto é, é possível saber sempre o estado atual.

Árvore de pesquisa

Para a concretização da formulação do problema é necessário representar a informação fornecida num grafo. Através da aplicação de algoritmos de procura informada e não-informada será possível encontrar os caminhos (um conjunto de pontos de recolha) desde a garagem de saída do transporte e o local de depósito de resíduos. Os vários nós do grafo serão pontos de recolha, que, no problema de pesquisa, representam os possíveis estados, e a ligação entre eles representará uma ligação física entre um ponto de recolha e outro, tendo em conta a orientação dessa ligação.

Com o enunciado foi fornecido um *dataset* reduzido e foi a partir deste que a base de conhecimento foi construída. Para tal, foi necessário desenvolver um programa para fazer o *parsing* do ficheiro. Na secção 4.1 dos **Anexos** é possível ver o código fonte do programa desenvolvido em *Python* utilizado para o efeito.

A tradução dos dados num grafo exigiu algumas decisões que serão de seguida enumeradas e explicadas.

No *dataset* existem várias linhas com valores de latitude e longitude iguais. Nessas linhas verificou-se que o nome da rua era semelhante, bem como o identificador do ponto de recolha, e que o único valor que varia diz respeito ao contentor. Deste modo, definiu-se que um **ponto de recolha** engloba a informação de todas as linhas que têm em comum o valor de latitude e longitude. Assim, um ponto de recolha terá a si associada uma **lista de contentores**. As capacidades e ocupações de contentores do mesmo tipo de resíduos serão somadas. Assim, por exemplo, as seguintes linhas do *dataset*

Latitude	Longitude	OBJECT	PONTO_RECOLH	PONTO_RECOLHA_LOCAL	CONTENTOR_RE	CONTENTC	CONTEN	CONT	CONTE
-9,143309	38,708079	355	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0090	90	1	90
-9,143309	38,708079	356	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0240	240	7	1680
-9,143309	38,708079	357	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Lixos	CV0090	90	1	90
-9,143309	38,708079	358	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Papel e Cartão	CV0240	240	6	1440
-9,143309	38,708079	359	Misericórdia	15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)	Papel e Cartão	CV0090	90	1	90

dão origem ao seguinte predicado na base de conhecimento:

```
ponto_recolha(
    15805,
    (-9.14330880914792, 38.7080787857025),
    'R do Alecrim ', par,
    [('Lixos', 1860, 1397), ('Papel e Cartao', 1530, 1465)]).
```

As ocupações foram geradas aleatoriamente e, como se pode verificar, embora existam 3 contentores do tipo *Lixo*, só aparece um contentor de tipo *Lixo* na lista de contentores do ponto de recolha, pois as capacidades e ocupações dos 3 contentores foram somadas.

Outras considerações importantes dizem respeito à ligação entre os pontos de recolha. Em primeiro lugar, os pontos de recolha foram organizados com base no seu identificador. De seguida, criou-se, para cada ponto de recolha, uma aresta que o liga ao primeiro ponto da rua que aparece imediatamente depois dos ':' na

coluna *PONTO_RECOLHA_LOCAL*. Para cada ponto de recolha foi também criada uma aresta com origem nesse ponto e destino no ponto de recolha com o identificador existente imediatamente a seguir.

Para o caso do primeiro ponto de recolha, com identificador 15805, serão criadas as seguintes arestas:

```
aresta(15805, 21944, 0.0004382025471219232).
aresta(15805, 15806, 6.897163411060243e-05).
```

A primeira aresta liga o ponto de recolha ao primeiro ponto de recolha da rua *R Ferragial*, que tem como identificador 21944. Já a segunda aresta liga o ponto ao ponto imediatamente abaixo na lista de pontos organizada com base no identificador. Ao predicado **aresta** foi também acrescentada a distância euclidiana entre os dois pontos de recolha.

Um último aspeto a realçar relaciona-se com as informações do tipo *Par*,*Impar* e *Ambos* que, apesar de terem sido incluídas no predicado *ponto_de_recolha*, não foram consideradas para a definição das arestas.

Em suma, foram definidos dois predicados para definição e pesquisa num grafo:

```
        ponto_de_recolha(Id,Lat,Lng,Rua,Dir,ListaLixos)
        aresta(Origem,Destino,Distancia)
```

2.2 Estratégias de procura

Para a resolução dos problemas propostos no enunciado, é necessária a implementação de algumas estratégias de pesquisa, que incluem estratégias de pesquisa não informada e informada.

2.2.1 Pesquisa não informada

Os algoritmos de procura não informada não possuem informação adicional sobre o caminho para o nó de destino para além daquelas disponíveis na definição do problema.

Pesquisa em profundidade

O algoritmo de pesquisa em profundidade começa a pesquisa no nó de origem e segue um caminho possível até ao nó final antes de iniciar outro caminho. Requer menos memória que a pesquisa em largura, pois apenas tem que guardar os nós do caminho que está a ser percorrido. Para além disso, para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que a procura em largura.

As propriedades deste algoritmo são:

- **Completo**: Não, é suscetível a entrar em ciclos e falha em espaços de profundidade infinita.
- **Ótimo**: Não.
- **Complexidade no tempo**: $O(b^m)$
- **Complexidade no espaço**: $O(bm)$

sendo $\boldsymbol{b}$ o número máximo de sucessores de um nó e $\boldsymbol{m}$ a máxima profundidade do espaço de estados.

No algoritmo implementado foram acrescentados mecanismos que impedem a ocorrência de ciclos.

Pesquisa em profundidade limitada

Um dos problemas da pesquisa em profundidade prende-se com a incapacidade desta lidar com caminhos infinitos e, deste modo, a pesquisa em profundidade com limite de profundidade procura evitar este problema fixando o nível máximo de procura.

O algoritmo da pesquisa em profundidade com profundidade limitada é muito semelhante ao algoritmo da pesquisa em profundidade, a única diferença é que os nós na profundidade $\boldsymbol{l}$ (o limite) não têm sucessores.

Pesquisa iterativa limitada em profundidade

A estratégia é executar uma pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade. Em geral é a melhor estratégia não informada para problemas com um grande espaço de pesquisa e em que a profundidade da solução não é conhecida.

Este algoritmo é também interessante no ponto de vista que, ao contrário da pesquisa em profundidade e pesquisa em profundidade com limite, encontra a solução ótima primeiro (se o custo for 1).

- **Completo**: Sim.
- **Ótimo**: Sim, se o custo for 1.
- **Complexidade no tempo**: $O(b^d)$
- **Complexidade no espaço**: $O(b^d)$

sendo $\boldsymbol{b}$ o número máximo de sucessores de um nó e $\boldsymbol{d}$ a profundidade da melhor solução.

Pesquisa em largura

Na pesquisa em profundidade todos os nós de menor profundidade são expandidos primeiro. A pesquisa deve ser iniciada num nó e, na travessia do grafo, todos os seus vizinhos devem ser visitados primeiro que os seus filhos.

- **Completo**: Sim
- **Ótimo**: Não. Só é ótimo se o custo de cada passo for 1.
- **Complexidade no tempo**: $O(b^d)$
- **Complexidade no espaço**: $O(b^d)$, guarda todos os nós em memória.

sendo $\boldsymbol{b}$ o número máximo de sucessores de um nó e $\boldsymbol{d}$ a profundidade da melhor solução.

2.2.2 Pesquisa informada

Nas estratégias de pesquisa informada são utilizadas heurísticas com o objetivo de fornecer "dicas"para a resolução do problema. Utilizando estimativas fornecidas por uma função de avaliação é possível escolher uma ordem específica de expansão dos nós.

Para o problema em mãos foi definida uma heurística simples: **a distância em linha reta de um ponto de recolha para o depósito**.

Foi escolhida esta heurística visto que é uma heurística otimista, ou seja, não sobrestima a verdadeira distância de um ponto a outro, o que é crucial para evitar que uma boa solução seja posta de parte.

```
estimativa(X,Y,Distancia) :-
  ponto_recolha(X,(Lat1,Lng1),_,_,_),
  ponto_recolha(Y,(Lat2,Lng2),_,_,_),
  Distancia is sqrt(((Lat1-Lat2)^2) + ((Lng1-Lng2)^2)).
```

Listing 2.1: Definição da estimativa da distância de um ponto a outro.

Os métodos heurísticos, no entanto, nem sempre fornecem a melhor solução, mas a sua grande vantagem está no facto de fornecem uma **boa solução** num espaço de **tempo razoável**. É importante destacar que mesmo que que seja encontrada uma solução ótima nestes algoritmos, esta não deixa de ser uma aproximação.

Pesquisa Gulosa

Na pesquisa Gulosa a estratégia é **expandir o nó que parece estar mais perto da solução**. Esta estimativa de proximidade de um nó a uma solução é feita com base na função de avaliação definida anteriormente, que determina a distância euclidiana entre dois nós. Resumidamente, pretende-se que, fazendo a escolha localmente ótima em cada fase, se consiga encontrar um ótimo global.

As propriedades da desta pesquisa são:

- **Ótima**: Não.
- **Completa**: Não, é suscetível a entrar em ciclos.
- **Complexidade no tempo**: $O(b^m)$
- **Complexidade no espaço**: Mantém todos os nós na memória, $O(b^m)$.

sendo $\boldsymbol{b}$ o número máximo de sucessores de um nó e $\boldsymbol{m}$ a máxima profundidade do espaço de estados.

O algoritmo implementado foi otimizado de modo a evitar ciclos.

Pesquisa A*

O algoritmo A* combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo estimado que falta até ao destino. Este algoritmo expande menos a árvore de pesquisa e determina uma solução, que é ótima, mais rapidamente.

As propriedades deste algoritmo são:

- **Ótimo**: Sim.
- **Completa**: Sim.
- **Complexidade no tempo**: $O(b^m)$
- **Complexidade no espaço**: Mantém todos os nós na memória, $O(b^m)$.

sendo $\boldsymbol{b}$ o número máximo de sucessores de um nó e $\boldsymbol{m}$ a máxima profundidade do espaço de estados.

2.3 Resolução dos requisitos obrigatórios

Para a resolução das várias alíneas, os algoritmos foram ligeiramente alterados de modo a permitir obter as informações necessárias. Analisemos, como exemplificação, o caso do algoritmo de procura em profundidade.

```
caminho_df(Garagem, Deposito,TipoLixo,[Garagem|Solucao],DistanciaTotal,NumeroPontosRecolha
    ,QuantidadeRecolhida):-
  resolvedf(Garagem,Deposito,TipoLixo,[Garagem],Solucao,DistanciaTotal,NumeroPontosRecolha
    ,QuantidadeRecolhida,0).
```

À medida que o caminho é calculado são também calculados valores como a **distância total percorrida**, **o número de pontos de recolha** e a **quantidade de lixo recolhida**.

```
resolvedf(Nodo,Deposito,TipoLixo,Historico,[ProxNodo|T],DistanciaTotal,NumeroPontosRecolha
    ,QuantidadeRecolhida,Acc) :-
  Acc >= 15000,
  adjacente(Nodo,ProxNodo,Distancia),
  nao(membro(ProxNodo,Historico)),
  ponto_recolha(Nodo,_,_,_,ListaLixos),
  temLixo(TipoLixo,ListaLixos,Ocupacao,PontoRecolha),
  resolvedf(ProxNodo,Deposito,TipoLixo,[ProxNodo|Historico],T,DistanciaTotalAcc,
    NumeroPontosRecolhaAcc,QuantidadeRecolhida,15000),
  DistanciaTotal is Distancia + DistanciaTotalAcc,
  NumeroPontosRecolha is PontoRecolha + NumeroPontosRecolhaAcc.

resolvedf(Nodo,Deposito,TipoLixo,Historico,[ProxNodo|T],DistanciaTotal,NumeroPontosRecolha
    ,QuantidadeRecolhida,Acc) :-
  Acc < 15000,
  adjacente(Nodo,ProxNodo,Distancia),
  nao(membro(ProxNodo,Historico)),
  ponto_recolha(Nodo,_,_,_,ListaLixos),
  temLixo(TipoLixo,ListaLixos,Ocupacao,PontoRecolha),
  Acc2 is Acc + Ocupacao,
  resolvedf(ProxNodo,Deposito,TipoLixo,[ProxNodo|Historico],T,DistanciaTotalAcc,
    NumeroPontosRecolhaAcc,QuantidadeRecolhida,Acc2),
  DistanciaTotal is Distancia + DistanciaTotalAcc,
  NumeroPontosRecolha is PontoRecolha + NumeroPontosRecolhaAcc.
```

Para o nó que está a ser visitado, verifica-se sempre se este contem o tipo de resíduos que foi passado como argumento. Se existir esse tipo de lixo, o número de pontos de recolha será incrementado.

No caso da capacidade máxima ter sido atingida, deixar-se-á de somar a capacidade do transporte atual à ocupação do contentor que está no nó que está a ser visitado atualmente.

Esta lógica, como já foi referido, foi implementada para todos os algoritmos. No caso dos algoritmos de pesquisa não informada, foi implementada na função que determina um nós adjacente:

```
expande_gulosa(Deposito,TipoLixo,Caminho,ExpandeCaminhos) :-
  findall(NovoCaminho, adjacente_g(Deposito,TipoLixo,Caminho,NovoCaminho), ExpandeCaminhos
    ).

adjacente_g(Deposito,TipoLixo,[Nodo|Caminho]/Dist/Pontos/Quant/_,[ProxNodo,Nodo|Caminho]/
    NovaDist/NovoPontos/NovaQuant/Est) :-
  Quant < 15000,
  adjacente(Nodo,ProxNodo,Distancia),
  nao(membro(ProxNodo,Caminho)),
  ponto_recolha(ProxNodo,(_,_),_,_,ListaLixos),
  temLixo(TipoLixo,ListaLixos,Ocupacao,Ponto),
  NovaDist is Dist+Distancia,
  NovaQuant is Quant+Ocupacao,
  NovoPontos is Pontos+Ponto,
  estimativa(ProxNodo,Deposito,Est).

adjacente_g(Deposito,TipoLixo,[Nodo|Caminho]/Dist/Pontos/Quant/_,[ProxNodo,Nodo|Caminho]/
    NovaDist/NovoPontos/15000/Est) :-
  Quant >= 15000,
  adjacente(Nodo,ProxNodo,Distancia),
  nao(membro(ProxNodo,Caminho)),
  ponto_recolha(ProxNodo,(_,_),_,_,ListaLixos),
  temLixo(TipoLixo,ListaLixos,Ocupacao,Ponto),
  NovaDist is Dist+Distancia,
  NovoPontos is Pontos+Ponto,
  estimativa(ProxNodo,Deposito,Est).
```

2.3.1 Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território

Com as alterações ao algoritmos referenciadas anteriormente apenas é possível obter informações acerca da distância percorrida, pontos de recolha visitados e quantidade recolhida pelo transporte para um dado tipo de resíduos. Para obrigar o transporte a passar por certos pontos de recolha foi necessária a adição de uma pequena verificação no algoritmo de pesquisa em profundidade e pesquisa em largura. Quando se tem uma possível solução "em mãos"é verificado se essa soluções inclui os pontos de recolha especificados e, caso contrário, o Prolog fará *backtracking* e tentará encontrar outra solução.

```
percorre_territorio(_,[]).

percorre_territorio(S,[Ponto|Territorio]) :-
  membro(Ponto,S),
  percorre_territorio(S,Territorio).

caminho_bf(Garagem, Deposito,TipoLixo,Territorio,Solucao,DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida):-
  resolvebf(Deposito,TipoLixo,[([Garagem],0,0,0)],InvSolucao,DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida),
  reverse(InvSolucao,Solucao),
  percorre_territorio(Solucao,Territorio).

  caminho_df(Garagem, Deposito,Territorio,TipoLixo,[Garagem|Solucao],DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida):-
  resolvedf(Garagem,Deposito,TipoLixo,[Garagem],Solucao,DistanciaTotal,NumeroPontosRecolha
    ,QuantidadeRecolhida,0),
  percorre_territorio([Garagem|Solucao],Territorio).
```

Algo importante de referir é que foi considerado que os 'Lixos' são resíduos indiferenciados e, portanto, para recolha indiferenciada aquilo que se deve colocar na parte do *TipoLixo* é, exatamente, 'Lixo'.

2.3.2 Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher)

Para identificar os circuitos com mais pontos de recolha podemos calcular todos os caminhos possíveis de uma dada origem para um dado destino e escolher aquele com mais pontos de recolha. Essa escolha torna-se fácil, pois à medida que os caminhos são construídos, vai sendo verificado se no ponto de recolha que está a ser visitado existe o tipo de lixo passado como argumento.

```
melhorCaminhoMaxPontosRecolhaDF(Garagem,Destino,TipoLixo,PontosRecolha,MelhorCaminho) :-
  findall((Solucao,PontosRecolha),caminho_df(Garagem,Destino,TipoLixo,Solucao,Distancia,
    PontosRecolha,QuantidadeRecolhida), Solucoes),
  max(Solucoes,(MelhorCaminho,PontosRecolha)).
```

```
?- melhorCaminhoMaxPontosRecolhaDF(15805,15843,'Lixos',Pontos,S);true.
Tempo: 6860
Pontos = 76,
S = [15805, 15806, 15807, 15808, 15809, 15810, 15811, 15812, 15813, 15814, 15815, 15818, 15819, 15820, 15821, 15822, 15823, 15824, 15825, 15827, 15828, 158
30, 15831, 15832, 15833, 15834, 15836, 15845, 15846, 15847, 15848, 15849, 15850, 15851, 15852, 15853, 15855, 15856, 15857, 15858, 15859, 15860, 15861, 1586
2, 15863, 15864, 15865, 15866, 15867, 15868, 15888, 15889, 15890, 15891, 15892, 15893, 15894, 15896, 15897, 15898, 15899, 15871, 15873, 15875, 15876, 15877
, 15878, 15879, 15880, 15881, 15882, 15883, 15884, 15885, 15886, 15842, 15843] ,
```

Figura 2.1: Caminho com origem em 15805 e destino em 15843 com mais pontos de recolha de Lixos.

```
?- melhorCaminhoMaxPontosRecolhaDF(15805,15843,'Vidro',Pontos,S);true.
Tempo: 6781
Pontos = 1,
S = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15888, 15889, 15811, 15812, 15813, 15819, 15820, 15821, 15822, 15823, 15824, 15825, 15827, 15828, 158
30, 15831, 15832, 15833, 15834, 15836, 15845, 15846, 15847, 15848, 15849, 15850, 15851, 15852, 15868, 15869, 15871, 15873, 15865, 15885, 15855, 15856, 1585
7, 15858, 15859, 15860, 15861, 15842, 15843] ,
```

Figura 2.2: Caminho com origem em 15805 e destino em 15843 com mais pontos de recolha de Vidro.

```
?- melhorCaminhoMaxPontosRecolhaDF(15805,15843,'Embalagens',Pontos,S);true.
Tempo: 5250
Pontos = 1,
S = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15888, 15889, 15811, 15812, 15813, 15819, 15820, 15821, 15822, 15823, 15824, 15834, 15836, 15845, 158
46, 15847, 15848, 15849, 15850, 15851, 15852, 15868, 15869, 15871, 15873, 15865, 15885, 15855, 15856, 15857, 15858, 15859, 15860, 15861, 15842, 15843] ■
```

Figura 2.3: Caminho com origem em 15805 e destino em 15843 com mais pontos de recolha de Embalagens.

O problema de utilizar um *findall* com a procura em largura reside no facto de esta exigir muito espaço para executar, logo, não é, de todo, indicada para calcular todos os caminhos possíveis.

Os algoritmos de procura informada não servem para este cálculo, já que a heurística utilizada penas se relaciona com a distância e não com a quantidade de pontos de recolha.

2.3.3 Comparar circuitos de recolha tendo em conta os indicadores de produtividade

Com as operações e informações que foram incluídas nos algoritmos é possível, facilmente, determinar os indicadores de produtividade.

Para o caso da quantidade recolhida temos:

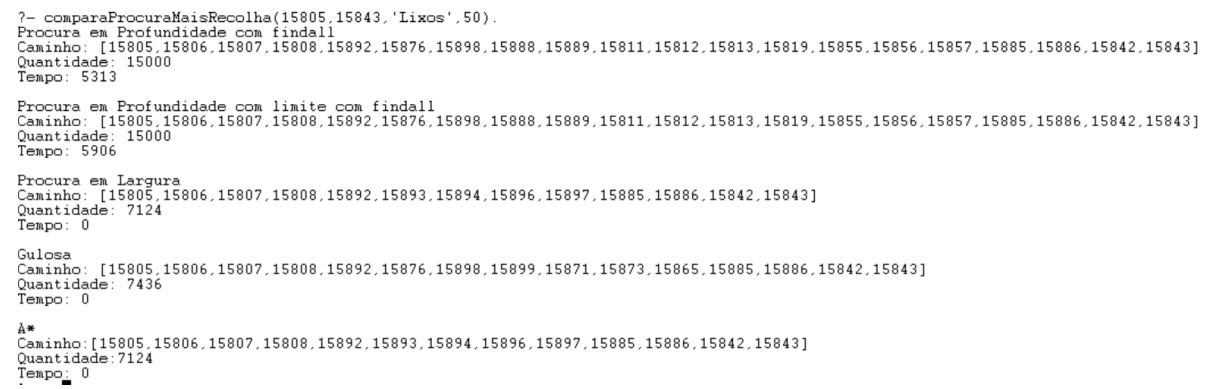

Figura 2.4: Comparação dos resultados dos diferentes algoritmos em termos da quantidade recolhida.

Já para o caso da distância média entre pontos de recolha temos:

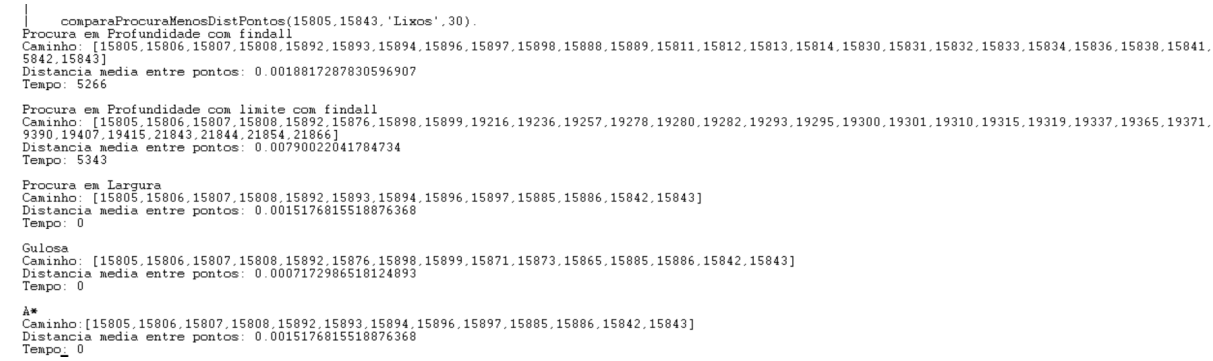

Figura 2.5: Comparação dos resultados dos diferentes algoritmos em termos da distância média entre pontos de recolha.

Para o caso do algoritmo de procura em profundidade e procura iterativa em profundidade com limite, foram definidas, para cada um, duas funções auxiliares que encontram todos os caminhos possíveis e selecionam aquele com o melhor valor tendo em conta o fator de produtividade.

```
1 melhorCaminhoMaxRecolhaDF(Garagem,Destino,TipoLixo,QuantidadeRecolhida, MelhorCaminho) :-
2   findall((Solucao,QuantidadeRecolhida),caminho_df(Garagem,Destino,TipoLixo,Solucao,_,_,
      QuantidadeRecolhida), Solucoes),
3   maxC(Solucoes,(MelhorCaminho,QuantidadeRecolhida)).
4
5 melhorCaminhoMaxRecolhaDFL(Garagem,Destino,TipoLixo,Limite,QuantidadeRecolhida,
      MelhorCaminho) :-
6   findall((Solucao,QuantidadeRecolhida),caminho_df_limite(Garagem,Destino,Limite,TipoLixo,
      Solucao,_,_,QuantidadeRecolhida), Solucoes),
7   maxC(Solucoes,(MelhorCaminho,QuantidadeRecolhida)).
8
9 melhorCaminhoMinDisEntrePontosDF(Garagem,Destino,TipoLixo,DistPontos,MelhorCaminho) :-
```

```
10   findall((Solucao,Distancia,PontosRecolha),caminho_df(Garagem,Destino,TipoLixo,Solucao,
       Distancia,PontosRecolha,_), Solucoes),
11   minDP(Solucoes,(MelhorCaminho,D,P)),
12   P=\=0, !,
13   DistPontos is D/P.
14
15 melhorCaminhoMinDisEntrePontosDFL(Garagem,Destino,TipoLixo,Limite,DistPontos,MelhorCaminho
       ) :-
16   findall((Solucao,Distancia,PontosRecolha),caminho_df_limite(Garagem,Destino,Limite,
       TipoLixo,Solucao,Distancia,PontosRecolha,_), Solucoes),
17   minDP(Solucoes,(MelhorCaminho,D,P)),
18   P=\=0, !,
19   DistPontos is D/P.
```

2.3.4 Escolher o circuito mais rápido (usando o critério da distância)

Utilizando o algoritmo de procura em profundidade é possível obter o caminho de menor comprimento, mas apenas se todos os caminhos possíveis forem calculados e comparados. Desse modo, foi definido um predicado com esse intuito:

```
1 melhorCaminhoMinDistanciaDF(Garagem,Destino,TipoLixo,Distancia, MelhorCaminho) :-
2   findall((Solucao,Distancia)
3   ,caminho_df(Garagem,Destino,TipoLixo,Solucao,Distancia,PontosRecolha,QuantidadeRecolhida
      ), Solucoes),
4   min(Solucoes,(MelhorCaminho,Distancia)).
```

```
?- melhorCaminhoMinDistanciaDF(15805,15843,'Lixos',Dist,Sol); true.
Dist = 0.010042181125374848,
Sol = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15899, 15871, 15873
, 15865, 15885, 15886, 15842, 15843] .
```

Figura 2.6: Caminho com menor distância.

Podemos observar os resultados obtidos com os algoritmos de procura informada e verificar que, embora não sejam a melhor solução, calcularam uma solução com um calor de distância total próxima e muito mais rapidamente que o anterior:

```
?- caminho_estrela(15805,15843,'Lixos',Sol,Dist,Pontos,Cap).
Sol = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15899, 15871, 15873
, 15865, 15885, 15886, 15842, 15843],
Dist = 0.01004218112537485,
Pontos = 14,
Cap = 7436 .

?- caminho_gulosa(15805,15843,'Lixos',Sol,Dist,Pontos,Cap).
Sol = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15899, 15871, 15873
, 15865, 15885, 15886, 15842, 15843],
Dist = 0.01004218112537485,
Pontos = 14,
Cap = 7436 .
```

Figura 2.7: Resultados dos algoritmos Gulosa e A* para o cálculo do percurso com menor distância

Comparando os tempos para diferentes pontos de origem e destino, obtemos:

	findall DF	**Gulosa**	**A***
15805-15843	4937ms	0ms	0ms
15831-15899	4266ms	0ms	0ms

2.3.5 Escolher o circuito mais eficiente (usando um critério de eficiência à escolha)

Considerou-se que um percurso **eficiente** é um percurso que permita que seja recolhida a quantidade máxima de resíduos que o transporte suporta, mas que, ao mesmo tempo, seja rápido. Se o critério fosse apenas escolher o caminho mais rápido o que aconteceria seria que o transporte chegaria ao depósito com muito espaço livre. Isso verifica-se executando, por exemplo, o algoritmo Gulosa e A*. O ideal é, efetivamente, uma vez a capacidade máxima atingida, o transporte escolheria, a partir do ponto de recolha em que se encontra, o caminho mais rápido para chegar ao depósito.

A implementação deste algoritmo baseia-se numa combinação entre a procura em profundidade e a gulosa. Primeiro é aplicada uma versão da procura em profundidade em que a pesquisa para quando a capacidade máxima é atingida e, de seguida, é aplicada a procura gulosa para obter o caminho aproximadamente mais rápido desse ponto onde a procura em profundidade parou, até à solução.

```
caminho_df_ef(Garagem, Deposito,TipoLixo,[Garagem|Solucao],DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida):-
  resolvedf_ef(Garagem,Deposito,TipoLixo,[Garagem],Solucao,DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida,0).

resolvedf_ef(_,_,_,_,[],0,0,Quant,Quant) :-
  Quant >= 15000, !.
resolvedf_ef(Deposito,Deposito,_,_,[],0,0,Acc,Acc).

resolvedf_ef(Nodo,Deposito,TipoLixo,Historico,[ProxNodo|T],DistanciaTotal,
    NumeroPontosRecolha,QuantidadeRecolhida,Acc) :-
  Acc < 15000,
  adjacente(Nodo,ProxNodo,Distancia),
  nao(membro(ProxNodo,Historico)),
  ponto_recolha(Nodo,_,_,_,ListaLixos),
  temLixo(TipoLixo,ListaLixos,Ocupacao,PontoRecolha),
  Acc2 is Acc + Ocupacao,
  resolvedf_ef(ProxNodo,Deposito,TipoLixo,[ProxNodo|Historico],T,DistanciaTotalAcc,
    NumeroPontosRecolhaAcc,QuantidadeRecolhida,Acc2),
  DistanciaTotal is Distancia + DistanciaTotalAcc,
  NumeroPontosRecolha is PontoRecolha + NumeroPontosRecolhaAcc.

caminho_ef(Garagem,Deposito,TipoLixo,Quant,MelhorCaminho) :-
  findall((Solucao,QuantidadeRecolhida),caminho_df(Garagem,Destino,TipoLixo,Solucao,
    Distancia,PontosRecolha,QuantidadeRecolhida), Solucoes),
  maxC(Solucoes,(MelhorCaminho,Quant)).


caminho_mais_eficiente(Garagem,Deposito,TipoLixo,Solucao,Quant) :-
    caminho_ef(Garagem,Deposito,TipoLixo,Quant,_SolDF),
    reverse(Solucao,[Fim|Inv]),
    caminho_gulosa(Fim,Deposito,TipoLixo,SolGulosa,_,_,QuantG),
    append(SolDF,SolGulosa,Solucao).
```

```
?- caminho_mais_eficiente(15805,15843,'Lixos',S,Q); true.
S = [15805, 15806, 15807, 15808, 15892, 15876, 15898, 15888, 15889, 15811, 15812, 15813, 15819,
 15855, 15856, 15857, 15857, 15885, 15886, 15842, 15843],
Q = 15000
```

Figura 2.8: Caminho mais eficiente.

Capítulo 3

Conclusão

A realização deste trabalho permitiu a consolidação de conceitos e aplicação das aprendizagens adquiridas ao longo do semestre. Foi um desafio programar os algoritmos de pesquisa em grafos, que já me eram familiares, numa linguagem como o Prolog. Ainda mais desafiante foi o facto de os ter de adaptar às questões do enunciado.

Algo muito importante de salientar é que foram realizados alguns testes em que a execução dos algoritmos Gulosa e A* não terminou, pois consumiram demasiada memória. Isso poderá se evitado utilizando estruturas de dados mais eficientes ou, até, fazendo alguns ajustes ao algoritmo.

Capítulo 4

Anexos

4.1 Programa para leitura do *Dataset*

```
dados = {}

def calculaDistancia(long1,long2,lat1,lat2):
    return math.sqrt(pow(lat2-lat1,2) + pow(long2-long1,2))


def leExcel():

    file = 'dataset.xlsx'
    wb_obj = openpyxl.load_workbook(file)
    sheet_obj = wb_obj.active

    m_row = sheet_obj.max_row

    for i in range(2, m_row + 1):
        latitude = sheet_obj.cell(row=i, column=1).value
        longitude = sheet_obj.cell(row=i, column=2).value
        ponto_recolha = sheet_obj.cell(row=i, column=5).value
        tipo_contentor = sheet_obj.cell(row=i, column=6).value
        contentor_total_litros = int(sheet_obj.cell(row=i, column=10).value)
        contentor_ocupacao = random.randint(1,contentor_total_litros)


        m = re.search(r'(\d+):( )*((\w| )+)',ponto_recolha)
        recolha_id = m.group(1)
        rua = m.group(3)

        m = re.search(r'\((\w+)(\s)*\(',ponto_recolha)
        if m is not None:
            sentido = m.group(1)
        else:
            sentido = ""

        m = re.search(r'\)\:(\s*)(((\w)| )+)(\-?)',ponto_recolha)
        if m is not None:
            ligacao_rua = m.group(2)
        else:
            ligacao_rua = ""
```

```
39
40          if (latitude,longitude,recolha_id) not in dados:
41              dados[(latitude,longitude,recolha_id)] = (rua,sentido,ligacao_rua,{})
42              dados[(latitude, longitude, recolha_id)][3][tipo_contentor] = [
    contentor_total_litros,contentor_ocupacao]
43
44          elif tipo_contentor not in dados[(latitude, longitude, recolha_id)][3]:
45              dados[(latitude, longitude, recolha_id)][3][tipo_contentor] = [
    contentor_total_litros,contentor_ocupacao]
46
47          elif tipo_contentor in dados[(latitude, longitude, recolha_id)][3]:
48              dados[(latitude, longitude, recolha_id)][3][tipo_contentor][0] +=
    contentor_total_litros
49              dados[(latitude, longitude, recolha_id)][3][tipo_contentor][1] +=
    contentor_ocupacao
50
51      #print(dados)
52
53  def criaGrafo():
54      f = open("grafo.pl", "a")
55
56      for key in dados.keys():
57          latitude = key[0]
58          longitude = key[1]
59          ponto = key[2]
60          direcao = dados[key][1]
61
62          if direcao=="":
63              direcao = "null"
64          else:
65              direcao = direcao.lower()
66
67
68          ponto_recolha = "ponto_recolha(" + ponto + ", (" + str(latitude) +", "+ \
69                          str(longitude) + "), '" +  dados[key][0] + "', " + direcao + ", ["
70          contentores = []
71
72          for contentor in dados[key][3].keys():
73              contentores.append("('" + contentor + "', " + str(dados[key][3][contentor][0])
74                                  + ", " + str(dados[key][3][contentor][1]) + ")")
75
76
77          for c in contentores:
78              if c == contentores[-1]:
79                  ponto_recolha = ponto_recolha + c + "]"
80              else :
81                  ponto_recolha = ponto_recolha + c + ", "
82
83          ponto_recolha = ponto_recolha + ")." + "\n"
84          f.write(ponto_recolha)
85
86
87      keys = dict(sorted(dados.items(), key= lambda x: x[0][2]))
88      for key in keys:
89          rua_conectada = dados[key][2]
90          id_conectada = 0
91
92          for keyC in keys:
93              if dados[keyC][0] in rua_conectada:
```

```
                id_conectada = keyC[2]
                break

        aresta1 = ""
        if id_conectada != 0:
            distancia = calculaDistancia(key[0],keyC[0],key[1],keyC[1])
            aresta1 = "aresta(" + key[2] + ", " + id_conectada + ", " + str(distancia) + "
    ).\n"
            f.write(aresta1)

        keys_list = list(keys)
        ultimo = keys_list[-1]
        if key[2] != ultimo[2]:
            next_key = keys_list[keys_list.index(key)+1]
            distancia = calculaDistancia(key[0],next_key[0],key[1],next_key[1])
            aresta2 = "aresta(" + key[2] + ", " + next_key[2] + ", " + str(distancia) + ")
    .\n"

            if (aresta2 != aresta1):
                f.write(aresta2)


    f.close()


if __name__ == '__main__':
    leExcel()
    criaGrafo()
```

Listing 4.1: Programa que faz *parsing* do *dataset* fornecido