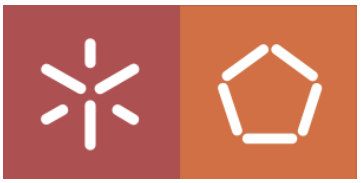

Universidade do Minho

Departamento de Informática

Sistemas de Representação de Conhecimento e Raciocínio

Avaliação Individual Versão Simplificada

Luís Manuel Pereira (a77667)

11 de junho de 2021

Resumo

O presente relatório descreve o trabalho prático realizado no âmbito da disciplina de *Sistemas de Representação de Conhecimento e Raciocínio*, ao longo do segundo semestre do terceiro ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Este instrumento de avaliação tem como tema os Métodos de Resolução de Problemas e de Procura.

O objetivo com a realização deste trabalho é aprofundar os conhecimentos relativos à formulação de problemas, a aplicação de diversas estratégias para a resolução de problema com o uso de algoritmos de procura e o desenvolvimento de mecanismos de raciocínio adequados a esta problemática.

O tema deste projeto foi o desenvolvimento de um sistema de recomendação de circuitos de recolha a partir dos circuitos de recolha de resíduos urbanos do concelho de Lisboa. Tendo sido disponibilizado um ficheiro dataset com os dados relativos aos pontos de recolha e informações sobre os seus respectivos contentores.

Conteúdo

Lista de Figuras

1 Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio* e tem como objetivo a avaliação individual sobre o tema de Métodos de Resolução de Problemas e de Procura.

Foi pedido ao aluno que desenvolva um sistema que permita importar os dados relativos aos diferentes circuitos, e representá-los numa base de conhecimento. Posteriormente, desenvolver um sistema de recomendação de circuitos de recolha a partir dos circuitos de recolha de resíduos urbanos do concelho de Lisboa.

Com o aprofundar dos conhecimentos relativos à língua de programação PROLOG e a Resolução de Problemas de Procura em mente foi desenvolvido um programa capaz de determinar diversos percursos entre locais a partir de diferentes condições de procura.

2 Estrutura

De seguida serão apresentados excertos dos ficheiros que formam a estrutura do projeto.

2.1 locais.pl

Ficheiro com o conhecimento dos locais. Cada local é representado pelo seu IdLocal, Latitude e Longitude. O local com IdLocal **15840** é utilizado como Garagem pelo que não apresenta contentores. Para um manuseamento mais eficiente dos dados foi utilizada a livraria **CSV** em **Python** para criar um parser capaz de estruturar e organizar os dados do ficheiro XML

```
import csv

f = open("/SRCR/locais.pl", "w")

with open('/SRCR/out.csv') as file:
    reader = csv.reader(file)
    prev = None
    value = None
    prevString = None
    for row in reader:

        if prev is not None and prev == row[4]:
            if value is not None:
                value = value + "," + prevString[2]
            else:
                value = prevString[2]
        else:
            if prevString is not None:
                f.write("local("+prevString[4]+","+ "["+ value + "," + prevString[2]+"
                ]"+","+prevString[0]+","+prevString[1]+",'"+prevString[3]+"').\n")
                value = row[2]
```

```
        prev = row[4]
        prevString = row
```

```
:- dynamic(local/4).
%--------------------------------- - - - - - - - - - - -  -  -  -  -   -
% Extenção do predicado local: IdLocal,Latitude, Longitude -> {V,F,D}
local(15840,-9.148518124,38.70980813). %Garagem

local(15841,-9.147044252,38.70801938).
local(15842,-9.146144218,38.70778398).
local(15843,-9.145639847,38.70788744).
local(15844,-9.145988636,38.70816366).
local(15845,-9.151993526,38.70762498).
local(15846,-9.150935625,38.70760718).
local(15847,-9.149935581,38.7076159).
```

2.2 contentores.pl

Ficheiro com o conhecimento dos Contentores. Cada Contentor é representado pelo seu IdContentor, IdLocal, Tipo Residuo, Tipo Contentor, Capacidade, QT e Litros.

```
import csv

with open('/SRCR/out.csv') as file:
    f = open("/SRCR/contentores.pl", "w")
    reader = csv.reader(file)
    prev = None
    value = None
    for row in reader:
        f.write("contentor("+row[2]+",'"+row[5]+"','"+row[6]+"',"+row[7]+","
        +row[8]+","+row[9]+").\n"  )
```

```
:- dynamic(contentor/7).
%--------------------------------- - - - - - - - - - - -  -  -  -  -   -
% Extenção do predicado contentor: IdContentor, IdLocal, Tipo Residuo, Tipo Contentor,
% Capacidade, QT, Litros  -> {V,F,D}
contentor(463,15841,lixos,cv0240,240,9,2160).
contentor(464,15841,lixos,cv0140,140,5,700).
contentor(465,15841,lixos,cv0240,240,9,2160).
contentor(466,15841,lixos,cv0140,140,5,700).
contentor(467,15842,lixos,cv0090,90,1,90).
contentor(468,15842,lixos,cv0090,90,1,90).
contentor(469,15843,lixos,cv0090,90,1,90).
contentor(470,15843,lixos,cv0140,140,4,560).
contentor(471,15843,lixos,cv0240,240,2,480).
contentor(472,15843,papel_cartao,cv0090,90,1,90).
contentor(473,15843,papel_cartao,cv0140,140,2,280).
contentor(474,15844,lixos,cv0140,140,1,140).
```

2.3 auxiliares.pl

Ficheiro que contem varios modules e predicados auxiliares ao projeto. Em destaque o module **statistics** do **SWI-Prolog** que foi utilizado como profiler para uma melhor análise de performance das várias estratégias de procura.

```
%distancia entre dois pontos
distancia(N1,N2,N3,N4,R):- N is sqrt((N3-N1)^2+(N4-N2)^2),N=R.

membro(X, [X|_]).
membro(X, [_|Xs]):-
        membro(X, Xs).

membros([], _).
membros([X|Xs], Members):-
        membro(X, Members),
        membros(Xs, Members).

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

inverso([], Xs, Xs).
inverso([X|Xs],Ys, Zs):-
        inverso(Xs, [X|Ys], Zs).

:- use_module(library(statistics)).
run_test(T,Distancia):-
  profile(pp_SolI(15840, 15843, 15847, T, Distancia, Quantidade)).
```

2.4 worker.pl

Ficheiro principal do projeto que contém as implementações dos diversos métodos de procura informada e não informada assim como predicados adaptados às funcionalidades pedidas.

```
%distancia entre Locais
distLocais(Id1_local, Id2_local,R):-
local(Id1_local,Lat1,Long1),
local(Id2_local,Lat2,Long2),
distancia(Lat1,Long1,Lat2,Long2,W), R is W.

%% Ver se é adjacente
adjacente(Id1_local, Id2_local,Distancia) :-
local(Id1_local,Lat1,Long1),
local(Id2_local,Lat2,Long2),
Dif is Id1_local-Id2_local,
Dif >= -2, Dif =< 2,
distLocais(Id1_local, Id2_local,R), Distancia is R.
```

```
seletiva(Id_local, Quantidade, Tipo) :-
findall(Quantidade2, contentor(_, Id_local, Tipo, _, _, Quantidade2, _), L),
sumList(L, Quantidade).

indiferenciada(Id_local, Quantidade) :-
findall(Quantidade2, contentor(_, Id_local, _, _, _, Quantidade2, _), L),
sumList(L, Quantidade).
```

3 Desenvolvimento do Projeto

Para iniciar o desenvolvimento do projeto foi primeiro necessário representar o conhecimento presente no ficheiro .xlsx . Utilizando uma script python foi possível converter para um conhecimento que o Prolog pudesse interpretar.

De seguida foi iniciado o desenvolvimento e teste dos predicados auxiliares aos requisitos do projeto (cálculo de distância, verificação de adjacência, membro e inverso da lista,etc).

Por fim foram criados os métodos de pesquisa informada e não informada para poder recolher as informações pretendidas aos percursos de recolha.

3.1 Auxiliares

Predicado *distancia/5* que baseado em 2 posições geométricas com uma latitude e longitude dá a distância entre elas.

```
distancia(N1,N2,N3,N4,R):- N is sqrt((N3-N1)^2+(N4-N2)^2),N=R.
```

Predicado *membro/2* testa se um elemento é membro de uma lista.

```
membro(X, [X|_]).
membro(X, [_|Xs]):-
        membro(X, Xs).

membros([], _).
membros([X|Xs], Members):-
        membro(X, Members),
        membros(Xs, Members).
```

Predicado *inverso/2* recebe uma lista e devolve o inverso dessa mesma lista.

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

inverso([], Xs, Xs).
inverso([X|Xs],Ys, Zs):-
        inverso(Xs, [X|Ys], Zs).
```

Predicado *remove_duplicates/2* recebe uma lista e devolve uma lista sem membros duplicados.

```
remove_duplicates([],[]).

remove_duplicates([H | T], List) :-
     member(H, T),
     remove_duplicates( T, List).

remove_duplicates([H | T], [H|T1]) :-
      \+member(H, T),
      remove_duplicates( T, T1).
```

Predicado *properList/3* que torna uma lista de tuplos com a informação de custo e percurso num valor total de custo e uma lista com o percurso.

```
properList([],0,[]).
properList([(Node,ProxNode,S)|Tail], Result,Lista2):-
properList(Tail,Time,Lista1),
Result is S + Time,
append([ProxNode],Lista1,Lista2).
```

Predicado *comprimento/2* para calcular o comprimento de uma lista.

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

Predicado *sumList/3* soma todos os elementos de uma lista.

```
sumList([], 0).
sumList([H|T], N):-
    sumList(T, X),
    N is X + H.
```

Predicado *atualizar/5* Atualiza o estado do histórico verificando se já se encontra ou não nesse mesmo histórico.

```
atualizar([], _, _, X-X).
atualizar([(_,Estado)|Ls], Vs, Historico, Xs-Ys) :-
        membro(Estado, Historico), !,
        atualizar(Ls, Vs, Historico, Xs-Ys).

atualizar([(Move, Estado)|Ls], Vs, Historico, [(Estado,[Move|Vs])|Xs]-Ys):-
        atualizar(Ls, Vs, Historico, Xs-Ys).
```

Predicado *seleciona/3* Seleciona outros caminhos de um histórico de caminhos.

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

Predicado *run_test/2* utiliza as ferramentas do SWI-Prolog para mostrar em gui uma análise do perfil de performance do predicado.

```
run_test(T,Distancia):-
  profile(pp_SolI(15840, 15843, 15847, T, Distancia, Quantidade)).
```

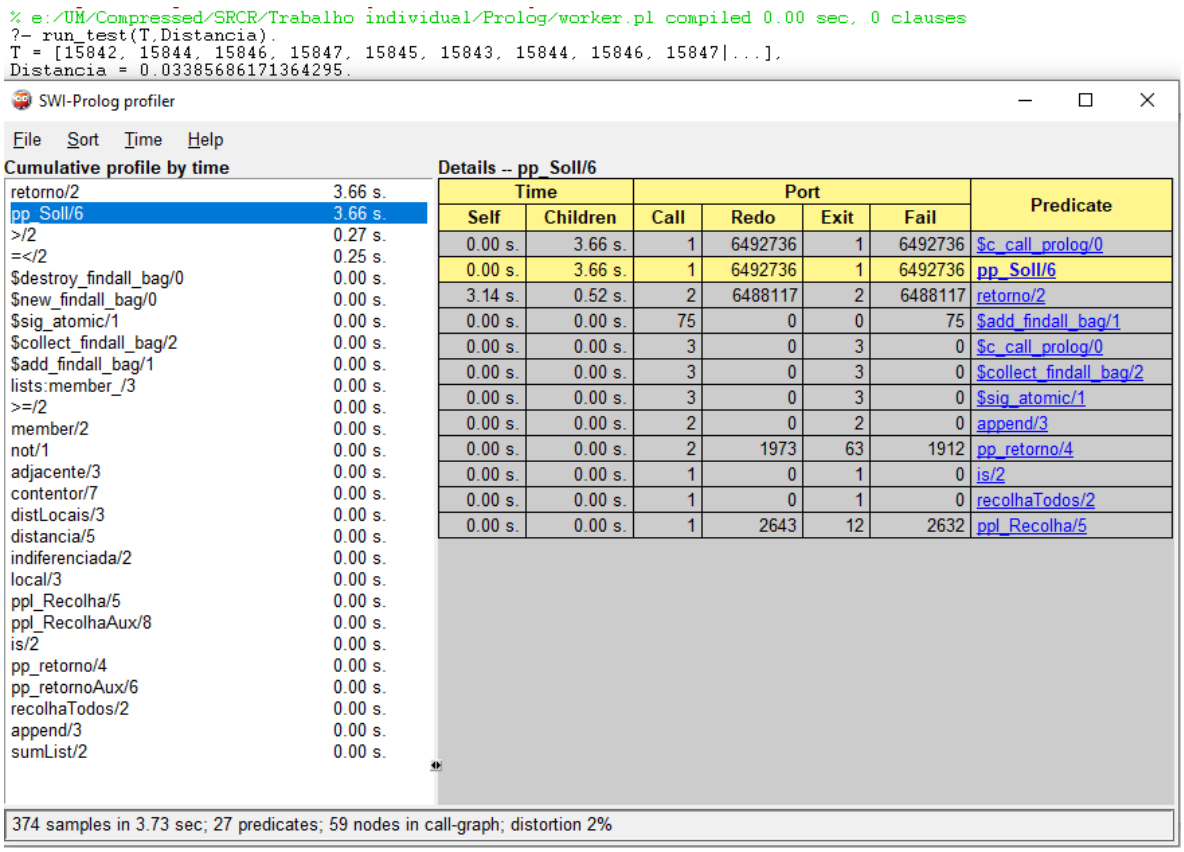

Figura 1: Exemplo da ferramenta de profilling do SWI-Prolog

Predicado *distLocais/3* devolve a distancia entre 2 locais pelo seu IdLocal.

```
distLocais(Id1_local, Id2_local,R):-
local(Id1_local,Lat1,Long1),
local(Id2_local,Lat2,Long2),
distancia(Lat1,Long1,Lat2,Long2,W), R is W.
```

```
?- distLocais(15840,15841,Distancia).
Distancia = 0.0023177414081102863.
```

Figura 2: Exemplo de utilização do predicado distLocais

Predicado *adjacente/3* verifica se dois locais sao adjacentes pelo seu IdLocal. No meu caso assumi que dois locais são adjacentes se os seus ids tiverem entre duas casas de espaçamento. Outro caso a considerar poderia ter sido um limite de distância entre locais mas considerando a precisão de distância que alguns locais podem ter entre si, tomei a decisão de usar o Id para uma melhor abstração nas soluções. Por fim é tambem calculada a distancia entre esses dois locais.

```
adjacente(Id1_local, Id2_local,Distancia) :-
local(Id1_local,Lat1,Long1),
local(Id2_local,Lat2,Long2),
Dif is Id1_local-Id2_local,
Dif >= -2, Dif =< 2,
distLocais(Id1_local, Id2_local,R), Distancia is R.
```

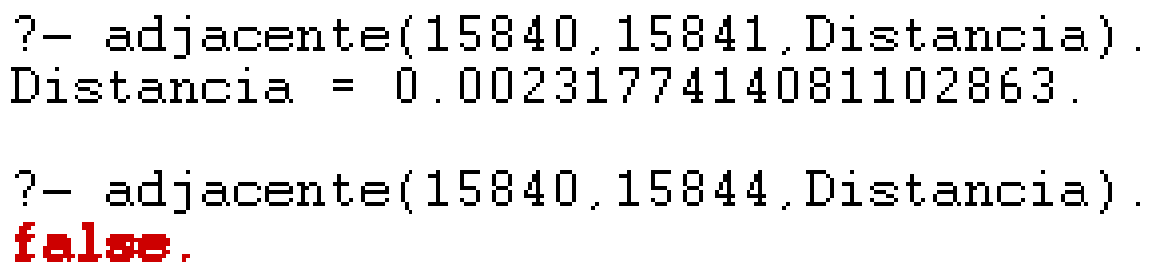

Figura 3: Exemplo de utilização do predicado adjacente

Predicado *seletiva/3* e predicado *indiferenciada/2*

De forma a respeitar o pedido foi necessário criar dois predicados a serem utilizados mais tarde. Um que será utilizado para uma pesquisa indiferenciada, não tendo em atenção o tipo de resíduo e outro predicado para uma pesquisa selectiva que tem em atenção o tipo de resíduo, devolvendo a quantidade desse resíduo.

```
seletiva(Id_local, Quantidade, Tipo) :-
findall(Quantidade2, contentor(_, Id_local, Tipo, _, _, Quantidade2, _), L),
sumList(L, Quantidade).

indiferenciada(Id_local, Quantidade) :-
findall(Quantidade2, contentor(_, Id_local, _, _, _, Quantidade2, _), L),
sumList(L, Quantidade).
```

```
?- indiferenciada(15843, Quantidade).
Quantidade = 10.

?- seletiva(15843, Quantidade, papel_cartao).
Quantidade = 3.
```

Figura 4: Exemplo de utilização do predicado seletiva e indiferenciada.

3.1.1 Tipos de recolha e retorno

De forma a responder a algumas das funcionalidades pedidas foi necessário a criação de diferentes métodos de recolha. Predicado *recolhaTodos/2* Predicado sem optimização ou preferência de caminho. Este predicado foi utilizado para retornar todos os percursos para poderem ser comparados.

Predicado *recolha/2* Este será o predicado normalmente utilizado com preferência por uma distância menor.

Predicado *recolhaQuant/2* Predicado utilizado para escolha de eficiência. Em que considerei eficiente a maior possível recolha de resíduos entre 2 locais.

Predicado *recolha2/2* Utilizado exclusivamente para verificar a maior distância e número de locais possível entre a garagem e o local de recolha..

Predicado *retorno/2* Predicado utilizado para todos os métodos de retorno baseado na menor distância possível. Esta consideração foi feita pois não existe recolha adicional de resíduo entre ponto de recolha e ponto de deposição e este mesmo com a garagem. Procurando ter o menor custo ao retornar da recolha.

```
recolhaTodos([(P,D,Q)],(P,D,Q)).
recolhaTodos([(_,D1,_)|L],(P2,D2,Q2)):-recolhaTodos(L,(P2,D2,Q2)).
recolhaTodos([(P1,D1,Q1)|L],(P1,D1,Q1)):-recolhaTodos(L,(_,D2,_)).

recolha([(P,D,Q)],(P,D,Q)).
recolha([(_,D1,_)|L],(P2,D2,Q2)):-recolha(L,(P2,D2,Q2)),D1 > D2.
```

```
recolha([(P1,D1,Q1)|L],(P1,D1,Q1)):-recolha(L,(_,D2,_)),D1 =< D2.

recolhaQuant([(P,D,Q)],(P,D,Q)).
recolhaQuant([(_,_,Q1)|L],(P2,D2,Q2)):-recolhaQuant(L,(P2,D2,Q2)),Q1 < Q2.
recolhaQuant([(P1,D1,Q1)|L],(P1,D1,Q1)):-recolhaQuant(L,(_,_,Q2)),Q1 >= Q2.

retorno([(P,D)],(P,D)).
retorno([(_,D1)|L],(P2,D2)):-retorno(L,(P2,D2)),D1>D2.
retorno([(P1,D1)|L],(P1,D1)):-retorno(L,(_,D2)),D1=<D2.

recolha2([(P,D,Q)],(P,D,Q)).
recolha2([(_,D1,_)|L],(P2,D2,Q2)):-recolha2(L,(P2,D2,Q2)),D1 < D2.
recolha2([(P1,D1,Q1)|L],(P1,D1,Q1)):-recolha2(L,(_,D2,_)),D1 >= D2.
```

4 Metodos de Pesquisa

4.1 Pesquisa não Informada

Para a pesquisa não informada foram implementados dois métodos de pesquisa diferentes, em profundidade, e em largura.

4.1.1 Pesquisa em Profundidade Indiferenciada

Na teoria, procura em profundidade é um algoritmo usado para realizar uma procura ou travessia numa árvore, estrutura de árvore ou grafo. Este algoritmo começa num nodo, e explora tanto quanto possível cada um dos seus ramos, antes de retroceder(backtracking).

A pesquisa foi separada em 3 partes. Inicialmente encontrando os percursos de recolha que satisfazem o tipo de recolha, neste caso utilizando a distância mais curta e devolvendo a quantidade de resíduo indiferenciada desse percurso.

Em segundo lugar temos o caminho do camião até ao local de disposição e finalmente o percurso do local de disposição até a garagem. Estes dois últimos métodos utilizam sempre a distância mais curta por razões explicitadas anteriormente nos tipos de recolha e retorno. Para criar o percurso e distância/custo final foram feitos os appends e somas necessárias. A distância/custo inclui o custo do caminho de retorno contrariamente à recolha.

O Predicado *pp_SolI/6* recebe o Id do local da garagem, Id do local de recolha e Id do local de deposição criando o percurso, a distancia e quantidade desse mesmo percurso.

```
% Profundidade Indiferenciada
pp_SolI(Garagem, Ponto, Deposicao, PercursoFinal, Distancia, Quantidade):-
findall((Percurso2,Distancia2,Quantidade2),ppI_Recolha(Garagem,Ponto,Percurso2,
Distancia2,Quantidade2),L1),
recolha(L1,(P1,D1,Quantidade)),
findall((Percurso3,Distancia3),
pp_retorno(Ponto,Deposicao,Percurso3,Distancia3),L2),
retorno(L2,(P2,D2)),
findall((Percurso4,Distancia4),
pp_retorno(Deposicao,Garagem,Percurso4,Distancia4),L3),
```

```
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
Distancia is D1+D2+D3.

ppI_Recolha(NodoIni, NodoFim, Percurso, Distancia, Quantidade) :-
ppI_RecolhaAux(NodoIni, NodoFim, [NodoIni], Percurso, 0, Distancia, 0, Quantidade).

ppI_RecolhaAux(NodoFim,NodoFim,_,[],Distancia,Distancia,Quantidade,Quantidade):-
Quantidade >= 1.

ppI_RecolhaAux(NodoIni,NodoFim,Hist,[ProxNodo|Percurso], InDist, OutDist, InQuant,
OutQuant) :-
adjacente(NodoIni,ProxNodo,Distancia),
indiferenciada(ProxNodo, ProxQuant),
not(member(ProxNodo,Hist)),
NDist is InDist + Distancia,
NQuant is InQuant+ProxQuant,
ppI_RecolhaAux(ProxNodo,NodoFim,[ProxNodo|Hist], Percurso, NDist, OutDist, NQuant,
OutQuant).

pp_retorno(NodoIni, NodoFim, Percurso, Distancia) :-
pp_retornoAux(NodoIni, NodoFim, [NodoIni], Percurso, 0, Distancia).

pp_retornoAux(NodoFim,NodoFim,_,[],Distancia,Distancia).
pp_retornoAux(NodoIni,NodoFim,Hist,[ProxNodo|Percurso], InDist, OutDist) :-
adjacente(NodoIni,ProxNodo,Distancia),
not(member(ProxNodo,Hist)),
NDist is InDist + Distancia,
pp_retornoAux(ProxNodo,NodoFim,[ProxNodo|Hist], Percurso, NDist, OutDist).


?- pp_SolI(15840, 15843, 15847, T, Distancia, Quantidade).
T = [15842, 15844, 15846, 15847, 15845, 15843, 15844, 15846, 15847|...],
Distancia = 0.03385686171364295,
Quantidade = 59 ,
```

Figura 5: Exemplo de utilização do predicado pp_SolI.

4.1.2 Pesquisa em Profundidade Seletiva

A pesquisa em profundidade seletiva funciona de maneira concordante com a indiferenciada, excepto o predicado auxiliar de recolha que utilizando o predicado seletiva limita a recolha ao tipo de resíduo.

O Predicado *pp_SolS/7* recebe o tipo de resíduo, Id do local da garagem, Id do local de recolha e Id do local de deposição criando o percurso, a distancia e quantidade desse mesmo percurso.

```
pp_SolS(Tipo, Garagem, Ponto, Deposicao, PercursoFinal, Distancia, Quantidade):-
findall((Per2,Dist2,Quant2),
ppS_Recolha(Tipo,Garagem,Ponto,Per2,Dist2,Quant2),L1),
recolha(L1,(P1,D1,Quantidade)),
findall((Per3,Dist3),
pp_retorno(Ponto,Deposicao,Per3,Dist3),L2),
retorno(L2,(P2,D2)),
append(P1, P2, Percurso),
findall((Per4,Dist4),
pp_retorno(Deposicao,Garagem,Per4,Dist4),L3),
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
Distancia is D1+D2+D3.


ppS_Recolha(Tipo,NodoIni, NodoFim, Percurso, Distancia, Quantidade) :-
ppS_RecolhaAux(Tipo,NodoIni, NodoFim, [NodoIni], Percurso, 0, Distancia, 0, Quantidade).

ppS_RecolhaAux(_,NodoFim,NodoFim,_,[],Distancia,Distancia,Quantidade,Quantidade):-Quantida
ppS_RecolhaAux(Tipo,NodoIni,NodoFim,Hist,[ProxNodo|Percurso], InDist, OutDist, InQuant, Ou
adjacente(NodoIni,ProxNodo,Distancia),
seletiva(ProxNodo, ProxQuant, Tipo),
not(member(ProxNodo,Hist)),
NDist is InDist + Distancia,
NQuant is InQuant + ProxQuant,
ppS_RecolhaAux(Tipo,ProxNodo,NodoFim,[ProxNodo|Hist], Percurso, NDist, OutDist, NQuant, Ou


?- pp_SolS(lixos,15840, 15843,15847, T, Distancia, Quantidade).
T = [15842, 15843, 15844, 15846, 15847, 15846, 15844, 15842, 15840],
Distancia = 0.019566080317190183,
Quantidade = 9 ;

?- pp_SolS(papel_cartao,15840, 15843,15847, T, Distancia, Quantidade).
T = [15842, 15843, 15844, 15846, 15847, 15846, 15844, 15842, 15840],
Distancia = 0.019566080317190183,
Quantidade = 3 ;
```

Figura 6: Exemplo de utilização do predicado pp_SolS.

4.1.3 Pesquisa em Largura

A pesquisa em largura começa por um vértice, neste caso a garagem. O algoritmo visita a garagem, depois visita todos os vizinhos, depois todos os vizinhos dos vizinhos, e assim por diante.

Com pensamento análogo à pesquisa em profundidade, a pesquisa em largura também foi dividida nas mesmas 3 partes.

```
bfs(Garagem, Destino, Deposicao, ListaFinal,CustoFinal) :-
breadthFirst([(Garagem, [])|Xs] - Xs, [], Destino, Caminho),
breadthFirst([(Destino, [])|Xs2] - Xs2, [], Deposicao, Caminho2),
breadthFirst([(Deposicao, [])|Xs3] - Xs3, [], Garagem, Caminho3),
properList(Caminho,Custo,Lista),
properList(Caminho2,Custo2,Lista2),
properList(Caminho3,Custo3,Lista3),
append(Lista,Lista2,CaminhoAux),
append(CaminhoAux,Lista3,ListaFinal),
CustoFinal is Custo+Custo2+Custo3.

breadthFirst(Garagem, Destino, Caminho) :-
    breadthFirst([(Garagem, [])|Xs] - Xs, [], Destino, Caminho).

breadthFirst([(Estado, Vs)|_] - _, _, Destino, Rs) :-
    Estado == Destino, !, inverso(Vs, Rs).

breadthFirst([(Estado, _)|Xs]-Ys, Historico, Destino, Caminho) :-
    membro(Estado, Historico), !,
    breadthFirst(Xs - Ys, Historico, Destino, Caminho).

breadthFirst([(Estado, Vs)|Xs] - Ys, Historico, Destino, Caminho) :-
    setof(((Estado, ProxNodo, Distancia), ProxNodo),
    adjacente(Estado, ProxNodo, Distancia), Ls),
    atualizar(Ls, Vs, [Estado|Historico], Ys-Zs),
    breadthFirst(Xs-Zs, [Estado|Historico], Destino, Caminho).
```

```
?- bfs(15840, 15847,15845, Caminho,Custo).
Caminho = [15841, 15843, 15845, 15847, 15845, 15843, 15841, 15840],
Custo = 0.024290786217757793.
```

Figura 7: Exemplo de utilização do predicado bfs.

4.1.4 Busca Iterativa Limitada em Profundidade, Seletiva e Indiferenciada.

Analogamente à pesquisa em profundidade e seguindo o mesmo raciocínio, exceto limitando o comprimento que um percurso pode ter. Este comprimento é explícito pelo utilizador e os percursos resultantes serão aqueles inferiores ao limite que neste caso chamo de N.

```
%--------------------------------- - - - - - - - - - - -  -  -  -  -  -
% Pesquisa Profundidade Indiferenciada Limitada
%--------------------------------- - - - - - - - - - - -  -  -  -  -  -

pp_SolIL(Garagem, Ponto, Deposicao, N, PercursoFinal, Distancia, Quantidade):-
findall((Percurso2,Distancia2,Quantidade2),ppI_Recolha(Garagem,Ponto,Percurso2,Distancia2,
recolhaTodos(L1,(P1,D1,Quantidade)),
findall((Percurso3,Distancia3),
pp_retorno(Ponto,Deposicao,Percurso3,Distancia3),L2),
retorno(L2,(P2,D2)),
append(P1, P2, Percurso),
findall((Per4,Dist4),
pp_retorno(Deposicao,Garagem,Per4,Dist4),L3),
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
comprimento(PercursoFinal, Tamanho),
        Tamanho < N,
Distancia is D1+D2+D3.

%--------------------------------- - - - - - - - - - - -  -  -  -  -  -
% Pesquisa Profundidade Seletiva Limitada
%--------------------------------- - - - - - - - - - - -  -  -  -  -  -

pp_SolSL(Tipo, Garagem, Ponto, Deposicao, N, PercursoFinal, Distancia, Quantidade):-
findall((Per2,Dist2,Quant2),ppS_Recolha(Tipo,Garagem,Ponto,Per2,Dist2,Quant2),L1),
recolhaTodos(L1,(P1,D1,Quantidade)),
findall((Per3,Dist3),
pp_retorno(Ponto,Deposicao,Per3,Dist3),L2),
retorno(L2,(P2,D2)),
append(P1, P2, Percurso),
findall((Per4,Dist4),
pp_retorno(Deposicao,Garagem,Per4,Dist4),L3),
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
comprimento(Percurso, Tamanho),
        Tamanho < N,
Distancia is D1+D2.
```

```
?- pp_SolIL(15840, 15843,15842, 5, T, Distancia, Quantidade).
T = [15842, 15843, 15842, 15840],
Distancia = 0.0072691729411111558,
Quantidade = 12 ;
false.

?- pp_SolIL(15840, 15843,15842, 6, T, Distancia, Quantidade).
T = [15842, 15843, 15842, 15840],
Distancia = 0.0072691729411111558,
Quantidade = 12 ;
T = [15841, 15842, 15843, 15842, 15840],
Distancia = 0.0073975094985729 51,
Quantidade = 40 ;
false.
```

Figura 8: Exemplo de utilização do predicado pp_SolIL.

4.2 Pesquisa Informada

Para a pesquisa informada foi implementado dois métodos de pesquisa muito semelhantes, o A* e o Gulosa(Greedy). Ambos estes métodos de pesquisa utilizam o critério de menor distância através do *adjacenteA/3*, que atualiza a estimativa com base no nodo inicial,final e atual. Para calcular a nova estimativa é calculada a distância entre o nodo atual e o próximo nodo, distância que será adicionada ao novo custo.

4.2.1 Pesquisa Gulosa

Algoritmo guloso ou Greedy é uma técnica de algoritmos para resolver problemas de otimização, realizando sempre a escolha que parece ser a melhor no momento. Fazendo uma escolha óptima local, na esperança de que esta escolha leva até a uma ótima solução global. Neste contexto, este algoritmo toma como variável para tomar decisões a distância de um nodo ao destino.

O algoritmo Greedy ou Guloso, foi implementado tendo em conta a heurıstica da distância entre os nodos adjacentes ao actual com o nodo final, escolhendo o que tem menor distância

```
gulosaR(Garagem, Ponto, Deposicao, PercursoFinal,CustoFinal):-
        aGulosa(Garagem,Ponto, Caminho1,Custo1),
        aGulosa(Ponto,Deposicao, Caminho2,Custo2),
        aGulosa(Deposicao,Garagem, Caminho3,Custo3),
        removehead(Caminho2,Caminho2S),
        removehead(Caminho3,Caminho3S),
        append(Caminho1, Caminho2S, PercursoMid),
        append(PercursoMid, Caminho3S, PercursoMid2),
        append(PercursoMid2, [Garagem], PercursoFinal),
        CustoFinal is Custo1+Custo2+Custo3.


aGulosa(Garagem,Deposicao, Caminho,Custo) :-
        distLocais(Garagem,Deposicao ,Estima),
```

```
        gulosa([[Garagem]/0/Estima], InvCaminho/Custo/_,Deposicao),
        removehead(InvCaminho,CaminhoInv),
        inverso(CaminhoInv, Caminho).

gulosa(Caminhos, Caminho, Deposicao) :-
        obtem_melhor(Caminhos, Caminho,Deposicao),
        Caminho = [Deposicao|_]/_/_.

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


obtem_melhor_g([Caminho], Caminho,SolucaoCaminho) :- !.

obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho,SolucaoCaminho
        Est1 =< Est2, !,
        obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho,SolucaoCaminho).

obtem_melhor_g([_|Caminhos], MelhorCaminho,SolucaoCaminho) :-
        obtem_melhor_g(Caminhos, MelhorCaminho,SolucaoCaminho).

expandeGulosa(Caminho, ExpCaminhos,SolucaoCaminho) :-
        findall(NovoCaminho, adjacenteA(Caminho,NovoCaminho,SolucaoCaminho), ExpCaminhos).


?- gulosaR(15840,15847, 15845, Caminho,Custo).
Caminho = [15840, 15841, 15842, 15844, 15846, 15846, 15846, 15844, 15842|...],
Custo = 0.02138938509557381 ,
```

Figura 9: Exemplo de utilização do predicado gulosaR.

4.2.2 Pesquisa A*

Algoritmo A* procura o caminho de um vértice inicial até um vértice final e a combinação de aproximações heurísticas semelhante ao algoritmo Breadth First (Procura em Largura) e do Algoritmo de Dijkstra. Neste contexto, este algoritmo tem como heurística a escolha da menor soma entre o nodo cuja distância do ponto onde estamos e a distância ao destino. Obtendo então a menor distância possível entre dois locais.

```
estrelaR(Garagem, Ponto, Deposicao, PercursoFinal,CustoFinal):-
        estrela(Garagem,Ponto, Caminho1,Custo1),
        estrela(Ponto,Deposicao, Caminho2,Custo2),
        estrela(Deposicao,Garagem, Caminho3,Custo3),
        removehead(Caminho2,Caminho2S),
```

```
        removehead(Caminho3,Caminho3S),
        append(Caminho1, Caminho2S, PercursoMid),
        append(PercursoMid, Caminho3S, PercursoFinal),
        CustoFinal is Custo1+Custo2+Custo3.

estrela(Garagem,Deposicao, Caminho,Custo) :-
        distLocais(Garagem,Deposicao ,Estima),
        aestrela([[Garagem]/0/Estima], InvCaminho/Custo/_,Deposicao),
        inverso(InvCaminho, Caminho).

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

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


obtem_melhor([Caminho], Caminho,SolucaoCaminho) :- !.

obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho,SolucaoCaminho)
        Custo1 + Est1 =< Custo2 + Est2, !,
        obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho,SolucaoCaminho).

obtem_melhor([_|Caminhos], MelhorCaminho,SolucaoCaminho) :-
        obtem_melhor(Caminhos, MelhorCaminho,SolucaoCaminho).

expande_aestrela(Caminho, ExpCaminhos,SolucaoCaminho) :-
        findall(NovoCaminho, adjacenteA(Caminho,NovoCaminho,SolucaoCaminho), ExpCaminhos).

adjacenteA([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, SolucaoCaminho)
        adjacente(Nodo, ProxNodo,Distancia),
        \+ member(ProxNodo, Caminho),
        NovoCusto is Custo + Distancia,
        distLocais(Nodo,ProxNodo,Est).


?- estrelaR(15840,15847, 15845, Caminho,Custo).
Caminho = [15840, 15842, 15844, 15846, 15847, 15846, 15845, 15846, 15844|...],
Custo = 0.02126104853811242 ,
```

Figura 10: Exemplo de utilização do predicado estrelaR.

5 Funcionalidades

5.1 Gerar os circuitos de recolha tanto indiferenciada como seletiva

Como esclarecido anteriormente os predicados pp_SolI pp_SolS geram os percursos tanto seletivos como indiferenciados.

```
?- pp_SolI(15840, 15843, 15847, T, Distancia, Quantidade).
T = [15842, 15844, 15846, 15847, 15845, 15843, 15844, 15846, 15847|...],
Distancia = 0.03385686171364295,
Quantidade = 59 .
```

Figura 11: Exemplo de geração de circuito indiferenciado.

```
?- pp_SolS(lixos,15840, 15843,15847, T, Distancia, Quantidade).
T = [15842, 15843, 15844, 15846, 15847, 15846, 15844, 15842, 15840],
Distancia = 0.019566080317190183,
Quantidade = 9 .

?- pp_SolS(papel_cartao,15840, 15843,15847, T, Distancia, Quantidade).
T = [15842, 15843, 15844, 15846, 15847, 15846, 15844, 15842, 15840],
Distancia = 0.019566080317190183,
Quantidade = 3 .
```

Figura 12: Exemplo de geração de circuito seletivo.

5.2 Identificar quais os circuitos com mais pontos de recolha

Foram criados novos predicados *pp_MaxS/7* e *pp_MaxI/6* com base na procura em profundidade. A recolha2 utilizada, em contrapartida da recolha normal dá-nos a maior distância possível e por sua vez o maior número de locais percorridos.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Circuito maior numero de pontos de recolha Selectivo
%-------------------------------- - - - - - - - - - -  -  -  -  -  -

pp_MaxS(Tipo, Garagem, Ponto, Deposicao, PercursoFinal, Distancia, Quantidade):-
findall((Per2,Dist2,Quant2),ppS_Recolha(Tipo,Garagem,Ponto,Per2,Dist2,Quant2),L1),
recolha2(L1,(P1,D1,Quantidade)),
findall((Per3,Dist3),pp_retorno(Deposicao,Garagem,Per3,Dist3),L2),
retorno(L2,(P2,D2)),
append(P1, P2, Percurso),
findall((Per4,Dist4),
pp_retorno(Deposicao,Garagem,Per4,Dist4),L3),
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
Distancia is D1+D2+D3.

%-------------------------------- - - - - - - - - - -  -  -  -  -  -
```

```
% Circuito maior numero de pontos de recolha Indiferenciado
%-------------------------------- - - - - - - - - - - - - - - - - -

pp_MaxI(Garagem, Ponto, Deposicao, PercursoFinal, Distancia, Quantidade):-
findall((Per2,Dist2,Quant2),ppI_Recolha(Garagem,Ponto,Per2,Dist2,Quant2),L1),
recolha2(L1,(P1,D1,Quantidade)),
findall((Per3,Dist3),pp_retorno(Deposicao,Garagem,Per3,Dist3),L2),
retorno(L2,(P2,D2)),
append(P1, P2, Percurso),
findall((Per4,Dist4),
pp_retorno(Deposicao,Garagem,Per4,Dist4),L3),
retorno(L3,(P3,D3)),
append(P1, P2, Percurso),
append(Percurso, P3, PercursoFinal),
Distancia is D1+D2+D3.

recolha2([(P,D,Q)],(P,D,Q)).
recolha2([(_,D1,_)|L],(P2,D2,Q2)):-recolha2(L,(P2,D2,Q2)),D1 < D2.
recolha2([(P1,D1,Q1)|L],(P1,D1,Q1)):-recolha2(L,(_,D2,_)),D1 >= D2.


?- pp_MaxI(15840, 15847, 15845, T, Distancia, Quantidade).
T = [15842, 15841, 15843, 15845, 15844, 15846, 15847, 15844, 15842|...],
Distancia = 0.04294505776776355,
Quantidade = 87 ,

?- pp_MaxS(lixos,15840, 15847, 15845, T, Distancia, Quantidade).
T = [15842, 15841, 15843, 15845, 15844, 15846, 15847, 15844, 15842|...],
Distancia = 0.04294505776776355,
Quantidade = 84 ,
```

Figura 13: Exemplo de circuitos com mais pontos de recolha.

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

Utilizando o predicado recolhaTodo, explícito anteriormente e aplicando a procura em profundidade é possível averiguar os vários percursos, as suas distâncias percorridas e quantidade de resíduos.

```
?- pp_SolI(15840, 15845, 15842, T, Distancia, Quantidade).
T = [15842, 15844, 15846, 15847, 15845, 15844, 15842, 15840],
Distancia = 0.021125307310725844,
Quantidade = 49 .

?- pp_SolI(15840, 15845, 15842, T, Distancia, Quantidade).
T = [15842, 15844, 15846, 15847, 15845, 15844, 15842, 15840],
Distancia = 0.021125307310725844,
Quantidade = 49 ;
T = [15842, 15844, 15846, 15845, 15844, 15842, 15840],
Distancia = 0.019125311001536263,
Quantidade = 30 ;
T = [15842, 15844, 15845, 15844, 15842, 15840],
Distancia = 0.019118074056033427,
Quantidade = 25 ;
T = [15842, 15844, 15845, 15844, 15842, 15840],
Distancia = 0.019118074056033427,
Quantidade = 25 ;
T = [15842, 15844, 15843, 15845, 15844, 15842, 15840],
Distancia = 0.019893085534089895,
Quantidade = 35 ;
T = [15842, 15844, 15843, 15845, 15844, 15842, 15840],
Distancia = 0.019893085534089895,
Quantidade = 35 ;
T = [15842, 15844, 15843, 15845, 15844, 15842, 15840],
Distancia = 0.019893085534089895,
Quantidade = 35 ;
T = [15842, 15844, 15843, 15845, 15844, 15842, 15840],
Distancia = 0.019893085534089895,
Quantidade = 35 
```

Figura 14: Exemplo de comparar circuitos de recolha.

5.4 Escolher o circuito mais rápido usando critério da distância

Como esclarecido anteriormente, a heuristica da distancia utilizada no algoritmo de procura A* da o circuito mais rapido.

```
?- estrelaR(15840,15847, 15845, Caminho,Custo).
Caminho = [15840, 15842, 15844, 15846, 15847, 15846, 15845, 15846, 15844|...],
Custo = 0.02126104853811242 .
```

Figura 15: Exemplo de circuito mais rapido.

5.5 Escolher o circuito mais eficiente

Tomando como critério de eficiência a maior quantidade possível de recolha de resíduos, foi criado um predicado recolhaQuant, explicado anteriormente que restringe os percursos aquele com maior quantidade possível.

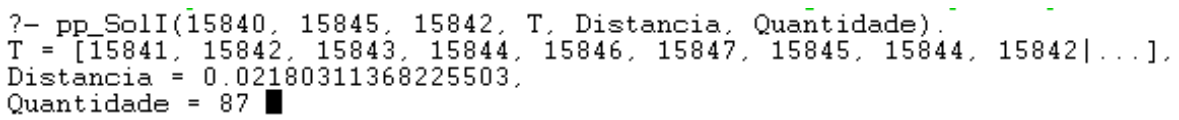

Figura 16: Exemplo de circuito mais eficiente.

6 Análise de performance e Resultados

A seguir apresento os resultados dos algorimos criados.

Specs: Ryzen 5 3600 4GHz, 16GB ddr4 3200MHz,CL16 Ram.

Os testes foram executados com a garagem 15940, ponto de recolha 15847, ponto de deposição 15845 e 14 conexões.

Estratégia	Tempo(s)	Espaço	Custo	Encontrou Melhor?
BFS	0.00	16MB	0.024290786217757793	Não
Greedy	72.11	6.35GB	0.02138938509557381	Não
A*	4.11	273MB	0.02126104853811242	Sim

Com 14 conexões o algoritmo BFS demora mais do que o tempo útil, pelo que diminui o número para 7. Por esta razão o algoritmo BFS não pode ser diretamente comparado com os outros.

Estratégia	Tempo(s)	Espaço	Custo	Encontrou Melhor?
DFS	38.17	16MB	0.021125307310725844	Não

7 Conclusão

Com a realização deste projeto individual consegui aprofundar os conhecimentos na linguagem PROLOG bem como um melhor entendimento dos diversos algoritmos de pesquisa informada e não informada pondo em prática os conceitos aprendidos nas aulas teóricas e nas aulas práticas.

Pode ser concluído que este é um tema de muita utilidade e aplicação em áreas distintas que permite a resolução de problemas de procura complexos.

8 Anexos

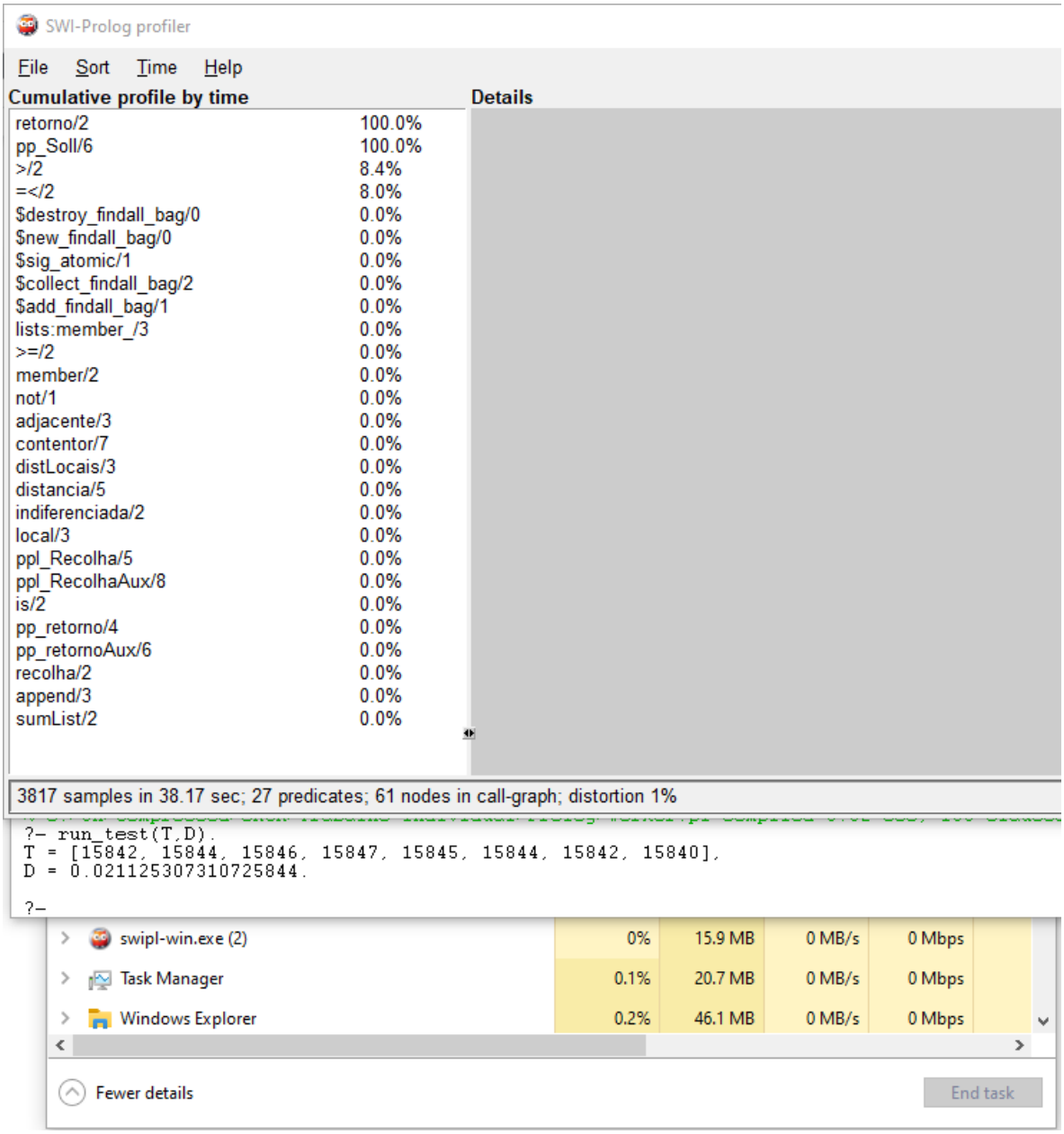

Figura 17: Profilling DFS

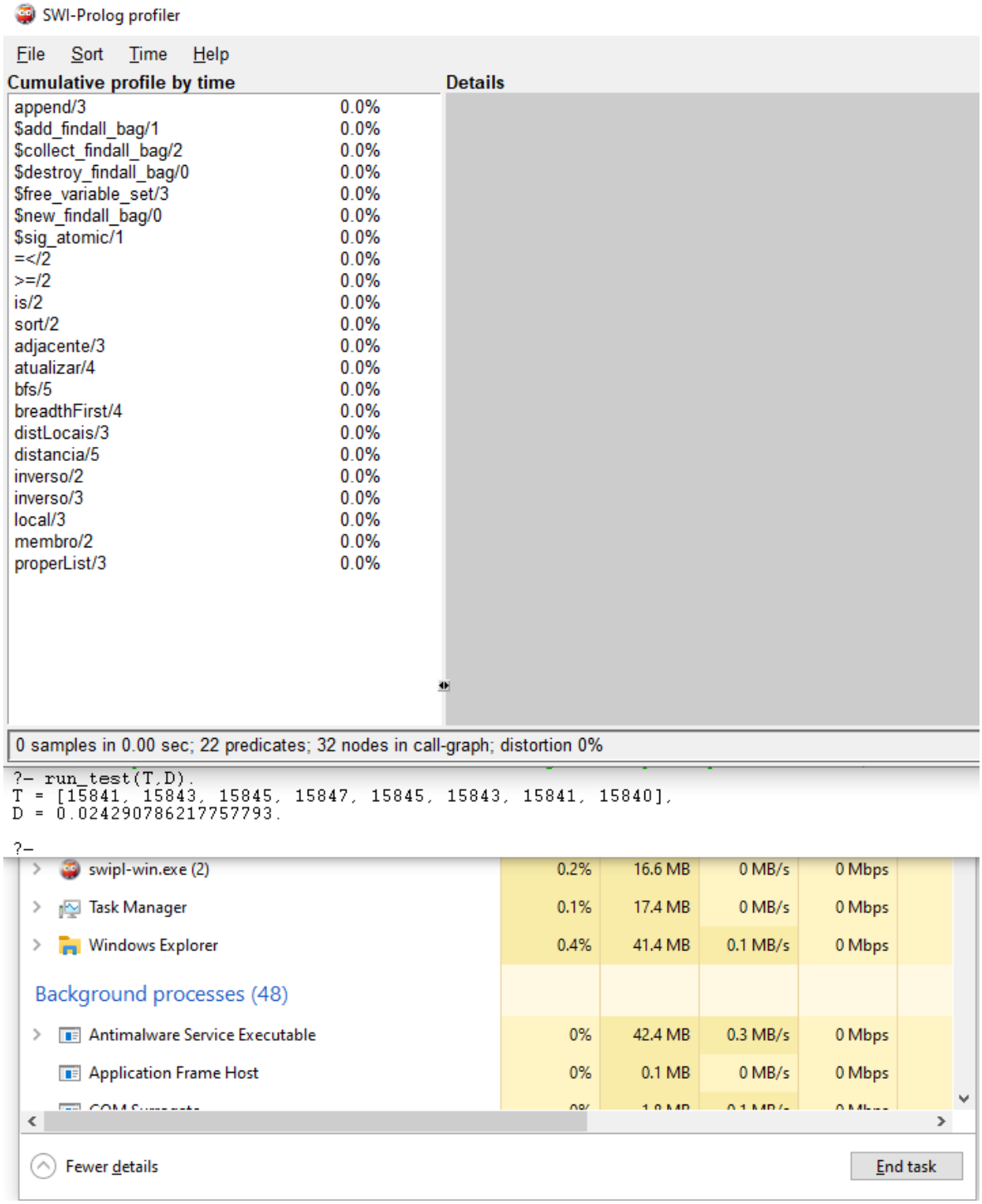

Figura 18: Profilling BFS

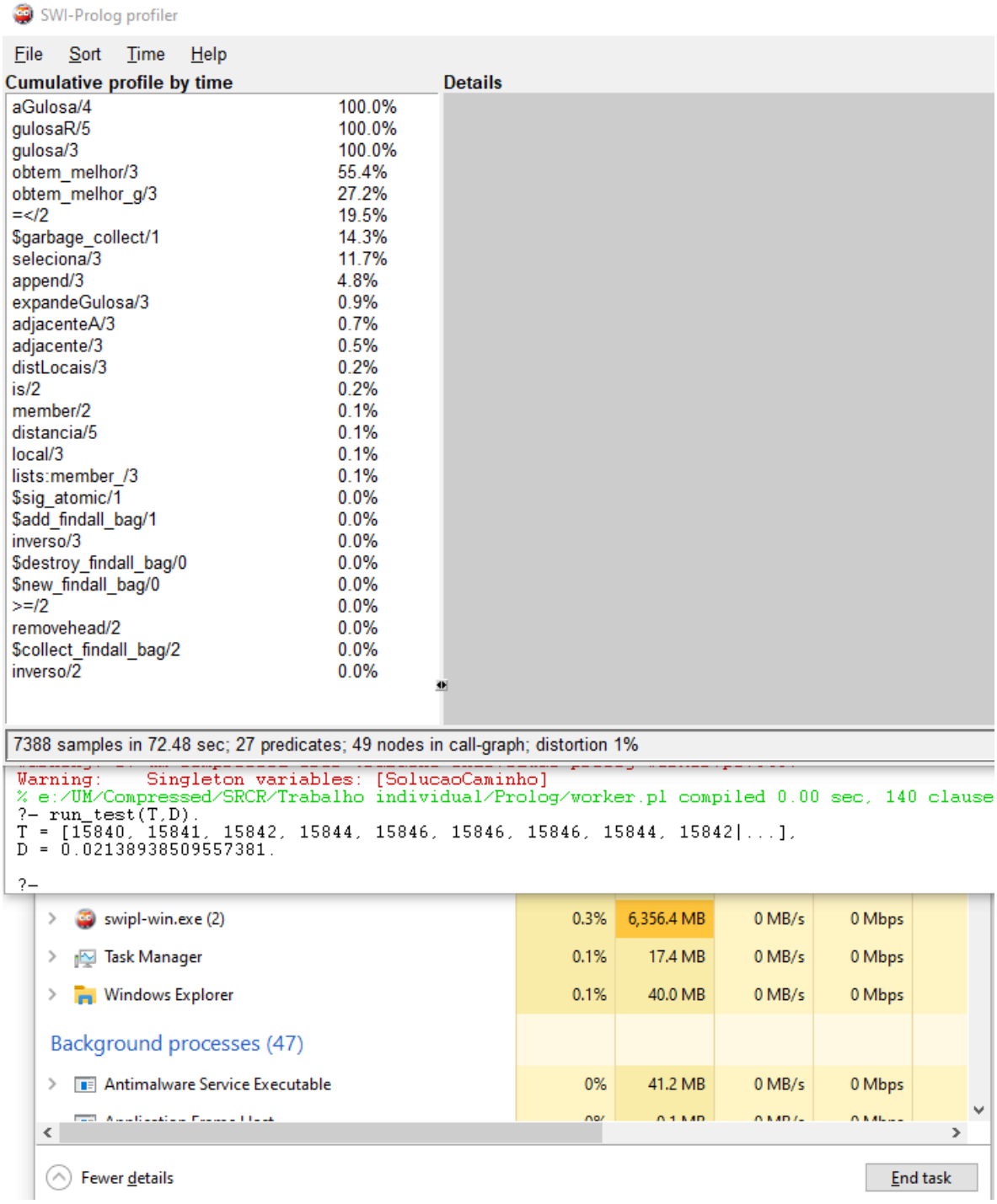

Figura 19: Profilling Greedy

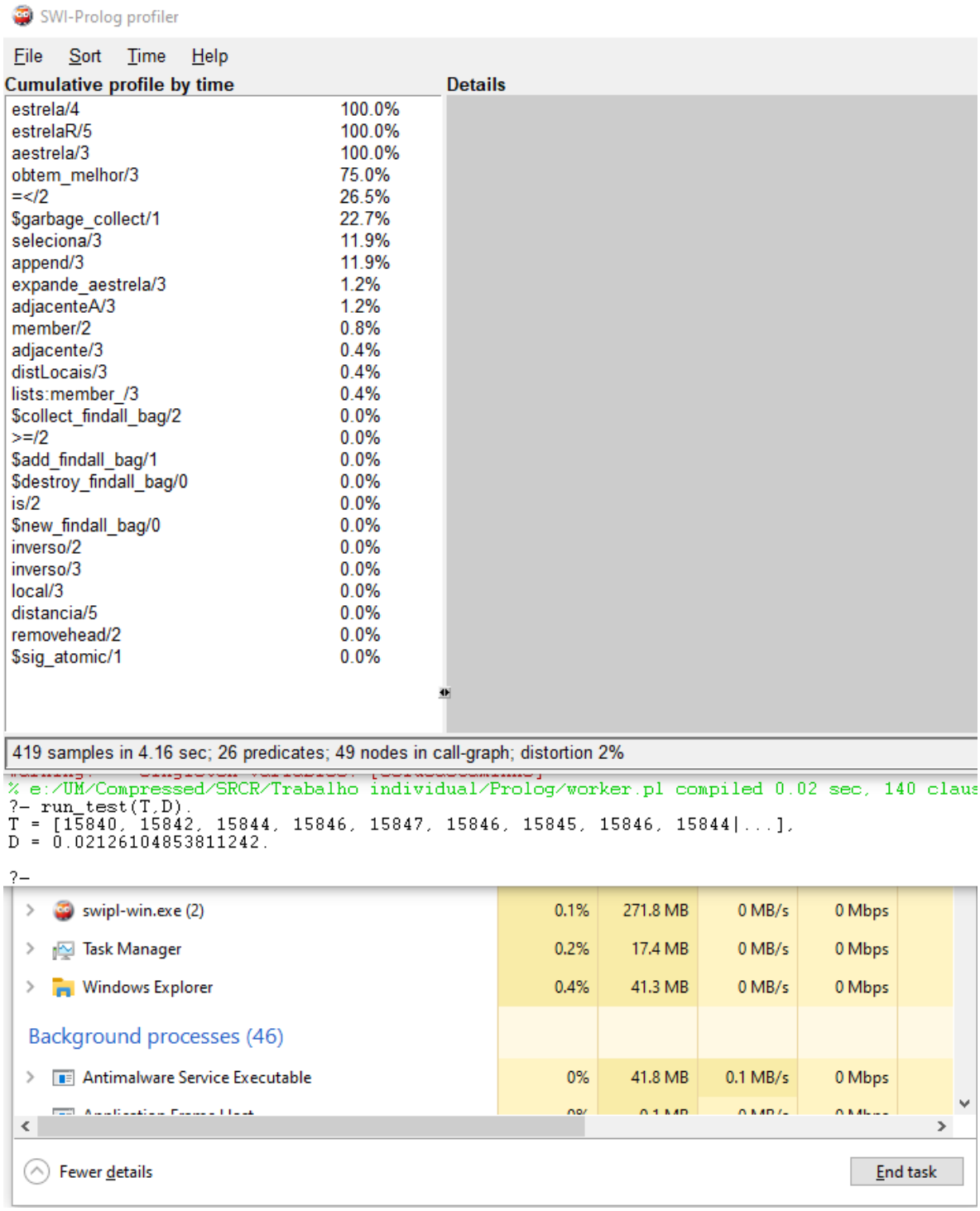

Figura 20: Profilling DFS A*