# Relatório de ALGAV


Turma 3DG

Grupo 41:

Mário António Afonso Cardoso 1201496

Miguel Corrêa Macedo 1201199

João Miranda 1201197

Ivo Oliveira 1190679

André Teixeira 1190384

# Representação do conhecimento de Domínio

Nós usamos os seguintes dados como base de conhecimento:

```
%idArmazem(<local>,<codigo>)
idArmazem('Arouca',1).
idArmazem('Espinho',2).
idArmazem('Gondomar',3).
idArmazem('Maia',4).
idArmazem('Matosinhos',5).
idArmazem('Oliveira de Azemeis',6).
idArmazem('Paredes',7).
idArmazem('Porto',8).
idArmazem('Povoa de Varzim',9).
idArmazem('Santa Maria da Feira',10).
idArmazem('Santo Tirso',11).
idArmazem('Sao Joao da Madeira',12).
idArmazem('Trofa',13).
idArmazem('Vale de Cambra',14).
idArmazem('Valongo',15).
idArmazem('Vila do Conde',16).
idArmazem('Vila Nova de Gaia',17).
```

*Figura 2-Armazéns*

```
entrega(4439, 20221205, 200, 1, 8, 10).
entrega(4438, 20221205, 150, 9, 7, 9).
entrega(4445, 20221205, 100, 3, 5, 7).
entrega(4443, 20221205, 120, 8, 6, 8).
entrega(4449, 20221205, 300, 11, 15, 20).
entrega(4398, 20221205, 310, 17, 16, 20).
entrega(4432, 20221205, 270, 14, 14, 18).
entrega(4437, 20221205, 180, 12, 9, 11).
entrega(4451, 20221205, 220, 6, 9, 12).
entrega(4452, 20221205, 390, 13, 21, 26).
entrega(4444, 20221205, 380, 2, 20, 25).
entrega(4455, 20221205, 280, 7, 14, 19).
entrega(4399, 20221205, 260, 15, 13, 18).
entrega(4454, 20221205, 350, 10, 18, 22).
entrega(4446, 20221205, 260, 4, 14, 17).
entrega(4456, 20221205, 330, 16, 17, 21).
```

*Figura 1-Entregas*

```
carateristicasCam(eTruck01,7500,4300,80,100,60).
```

*Figura 3-Camião*

```
dadosCam_t_e_ta(eTruck01,1,2,122,42,0).
dadosCam_t_e_ta(eTruck01,1,3,122,46,0).
dadosCam_t_e_ta(eTruck01,1,4,151,54,25).
dadosCam_t_e_ta(eTruck01,1,5,147,52,25).
dadosCam_t_e_ta(eTruck01,1,6,74,24,0).
dadosCam_t_e_ta(eTruck01,1,7,116,35,0).
dadosCam_t_e_ta(eTruck01,1,8,141,46,0).
dadosCam_t_e_ta(eTruck01,1,9,185,74,53).
dadosCam_t_e_ta(eTruck01,1,10,97,30,0).
dadosCam_t_e_ta(eTruck01,1,11,164,64,40).
dadosCam_t_e_ta(eTruck01,1,12,76,23,0).
dadosCam_t_e_ta(eTruck01,1,13,174,66,45).
dadosCam_t_e_ta(eTruck01,1,14,59,18,0).
dadosCam_t_e_ta(eTruck01,1,15,132,51,24).
dadosCam_t_e_ta(eTruck01,1,16,181,68,45).
dadosCam_t_e_ta(eTruck01,1,17,128,45,0).
```

*Figura 4-Dados Camião no percurso entre Armazéns*

Nota: A figura 4 contém apenas um excerto (apenas de quando o camião parte do armazém para outros) da totalidade.

# Planeamento de Entregas de Mercadorias com Camões Elétricos

- Obtenção da solução ótima para o Planeamento de Entrega de Mercadorias com 1 camião elétrico

A implementação do algoritmo para obter a solução mais rápida foi dividida em 3 fases:

1. Implementar parte de gerar todas listas possíveis através de permutações;
2. Implementar algoritmo que obtém a solução mais rápida sem ter em conta a bateria e o peso do camião (considerar o Camião sempre com Carga Total e Peso Máximo);
3. Implementar carga/perda de bateria e aumento/diminuição do peso no algoritmo.

O algoritmo final ficou da seguinte forma:

```
seq_custo_min(LC,Final):-get_time(Ti),(run;true),custo_min(LC,Final),get_time(Tf),TSol is Tf-Ti,write("Tempo final:"),write(TSol), nl.

run:-
    retractall(custo_min(_,_)),
    assertz(custo_min(_,100000)),
    armazens_entregas(LC),
    permutation(LC,LCPerm),
    add_matosinhos(LCPerm,LCPerm1),
    calcula_tempo(LCPerm1,Final),
    atualiza(LCPerm1,Final),
    fail.

atualiza(LCPerm1,Final):-
  custo_min(_,FinalMin),
  ((Final<FinalMin,!,retract(custo_min(_,_)),assertz(custo_min(LCPerm1,Final)),
    write(Final),nl)
    ;true).
```

*Figura 5-"seq_custo_min"*

Na consola colocamos seq_custo_min(X,Y). e este irá fazer o run até percorrer todas as listas e depois irá retornar o trajeto de menor tempo em X e o valor desse tempo em Y.

```
?- seq_custo_min(X,Y).
X = [5, 9, 3, 1, 5],
Y = 454.012711864406 8.
```

*Figura 6-Exemplo de uso do "seq_custo_min"*

O run faz não só as várias permutações da lista, mas também lhes adiciona Matosinhos ao início e fim (pois o Camião inicia e termina viagem sempre em Matosinhos) através add_matosinhos/2, calcula o tempo que cada lista demora a fazer todas a entregas com o calcula_tempo/2  e vê se é o menor tempo até ao momento através do atualiza/2, caso seja, atualiza a variável que contendo o menor tempo para esse valor.

```
add_matosinhos(L,LB):-append([5],L,LA),append(LA,[5],LB).
```

*Figura 7- "add_matosinhos"*

```
atualiza(LCPerm1,Final):-
  custo_min(_,FinalMin),
  ((Final<FinalMin,!,retract(custo_min(_,_)),assertz(custo_min(LCPerm1,Final)),
    write(Final),nl)
    ;true).
```

*Figura 8-"atualiza"*

O calculo_tempo/2 recebe uma lista com os armazéns onde vão ser colocadas as entregas e devolve o tempo que demora a fazer a colocação das entregas e voltar até Matosinhos.

Este predicado calcula o tempo de recarga de baterias do camião quando ele não tem energia suficiente para chegar a outro armazém e então tem de carregar até 80% no armazém em que está através do tempo_recarga/6, calcula o tempo extra para cargas intermédias através do tempo_extra/2 , calcula o tempo de colocação e remoção de entregas do camião com o tempo_entregas/2, calcula o tempo de viagem entre armazéns pelo predicado "tempo" e por fim soma todos estes tempos com o "soma_tudo".

```
tempo([_],0,_,_).
tempo([C1,C2|LC],Custo,PesoC,Nome):-
    entregas_peso(C1,LE),
    peso_entregas(LE,PesoE),
    PesoC1 is PesoC - PesoE,
    tempo([C2|LC],Custo1,PesoC1,Nome),
    (dadosCam_t_e_ta(_,C1,C2,Tempo,_,_)
    ;dadosCam_t_e_ta(_,C2,C1,Tempo,_,_)),
    peso_camiao(Nome,PesoB),
    Tempo1 is (PesoC1*Tempo)/PesoB,
    Custo is Custo1+Tempo1.
```

*Figura 9-"tempo"*

```
tempo_entregas([],0).
tempo_entregas([HC|LC],TempoE):-
        tempo_entregas(LC,TempoE1),
        buscar_entregas(HC,LI),
        entregas_tempo(LI,TempoT),
        TempoE is TempoE1 + TempoT.
```

*Figura 10-"tempo_entregas"*

```
tempo_extra([_],0).
tempo_extra([C1,C2|LC],ExtraC):-
        tempo_extra([C2|LC],ExtraC1),
        (dadosCam_t_e_ta(_,C1,C2,_,_,Extra);dadosCam_t_e_ta(_,C2,C1,_,_,Extra)),
        ExtraC is ExtraC1 + Extra.
```

*Figura 11-"tempo_extra"*

```
tempo_recarga([_],_,_,0,_,_).
tempo_recarga([C1,C2|LC],CarM,TempR,RecargaC,PesoC,Nome):-
        entregas_peso(C1,LE),
        peso_entregas(LE,PesoE),
        PesoC1 is PesoC - PesoE,
        tempo_recarga([C2|LC],CarM,TempR,RecargaC1,PesoC1,Nome),
        (dadosCam_t_e_ta(_,C1,C2,_,Energia,_);
        dadosCam_t_e_ta(_,C2,C1,_,Energia,_)),
        peso_camiao(Nome,PesoB),
        Energia1 is (PesoC1*Energia)/PesoB,
        ((CarM - Energia1 < 20, RecargaC is RecargaC1 + ((80 - (CarM - Energia1))*TempR)/60);RecargaC is RecargaC1 + 0).
```

*Figura 12-"tempo_recarga"*

- Estudo da complexidade do problema e da viabilidade de encontrar a solução ótima através da geração de todas as soluções

*Complexidade*

Começamos por registar numa tabela os tempos de execução do algoritmo para diferentes números de Armazéns de Entrega, de forma a sabermos quanto tempo demora a obter a melhor solução para os vários casos listados e percebermos o quanto o aumento de Armazéns influencia o tempo que demora a obter a solução.

| Nº de Armazéns de Entrega | Nº de Soluções | Lista com o trajeto | Tempo para fazer as entregas | Tempo de geração da solução |
|---|---|---|---|---|
| 3 | 6 | [5, 9, 3, 1, 5] | 454.01 m | 0.06s |
| 4 | 24 | [5, 9, 3, 1, 8, 5] | 464.87 m | 0.11s |
| 5 | 120 | [5, 9, 11, 3, 1, 8, 5] | 528.65 m | 0.55s |
| 6 | 720 | [5, 17, 1, 3, 8, 11, 9, 5] | 576.39 m | 4.47s |
| 7 | 5040 | [5, 17, 14, 1, 3, 8, 11, 9, 5] | 616.41 m | 101.50s |
| 8 | 40320 | [5,17,14,1,12,3,8,11,9,5] | 653.91 m | 2168.63 s |

Podemos concluir através dos dados da tabela que à medida que aumentamos o número de armazéns, o tempo para encontrar a solução aumenta exponencialmente, isto acontece, pois, quanto maior o número de armazéns maior será a lista que contém os mesmos e portanto existirão mais possibilidades no que toca a casos possíveis que a lista pode tomar(permutações).

*Viabilidade*

Podemos também ver que para 8 armazéns a solução demora muito tempo, o que faz que a partir deste valor ela não seja mais uma forma viável de se obter a solução mais *time-efficient*.

Logo, pode dizer-se que a partir de viagens que precisem de passar em 7 armazéns, é melhor recorrer a heurísticas.

- Minorante e Majorante

Já vimos que temos 4 parcelas que vão implicar no tempo final obtido:
• O tempo para ir de um armazém para o outro
• O tempo de descarga da entrega
• O tempo de carga das baterias no armazém (feito em paralelo com anterior)
• O tempo de carga das baterias durante um trajeto

## Minorante

Primeiramente vamos obter o conjunto de entregas que nos é fornecido e pôr para uma lista. Com esta lista, começamos um processo recursivo para fazer os cálculos de minorante para cada um dos elementos da lista, e fazer a soma de todos para uma varável que será o resultado.

```
%ve menor valor de tempo
recursao([],0):-!.
recursao([H|T],Min):-recursao(T,Min1),viagensParaId(H,Z),Min is Min1+Z.

minorante(M):-todosIdsDasEntregas(X),recursao(X,M).
%Minorante
```

*Figura 13 - Recursão para iterar pela lista*

Para cada entrega, obtemos o armazém-destino e depois íamos aos dados de camião ver todas as rotas que iam para esse armazém e metemos todos os tempos para uma lista.
Como é minorante vamos assumir que a carga que o camião leva é apenas a necessária para essa entrega, então é direto, chamamos o predicado “menorTempoParaID” para obter o menor tempo da lista.
Esse tempo seria para o camião cheio, pelo que temos de fazer as contas contando com apenas a carga necessária para essa entrega, pelo que fazemos as contas disponibilizadas para obter o tempo final da parcela “Trajeto”.

Para os tempos de descarga e carregamento das baterias, apenas se assume o tempo das descargas, então adicionamos somente o tempo da descarga nesse armazém.

A parcela de tempo “cargas adicionais” não entra para o minorante, visto que se assume o valor 0.

```
%viagensQvaoparaOId
viagensParaId(X,Z):-(findall(D,dadosCam_t_e_ta(_,_,X,D,_,_),R)), menorTempoParaId(R,Y),entrega(_,_,U,X,_,K), contas(U,Y,P), Z is P+K.
```

*Figura 14 - Obter minorante para cada entrega*

Obtemos assim o minorante para uma entrega, e como foi referido anteriormente o predicado de recursão vai fazer este procedimento para todas as entregas do conjunto e somá-las, pelo que se obtém o minorante final.

```
?- minorante(M).
M = 203.564406779661.
```

*Figura 15 – Minorante*

## Majorante

Para o majorante a primeira parte é igual, vamos obter o conjunto de entregas que nos é fornecido e pôr para uma lista. Com esta lista, começamos um processo recursivo para fazer os cálculos de majorante para cada um dos elementos da lista, e fazer a soma de todos para uma varável que será o resultado.

```
majorante(M):-todosIdsDasEntregas(X),
recursaoMaj(X,MedioTrajeto),
recursaoTempoCarregar(X,MedioCarregar),
contasTempo(MedioCarregar,R),
M is MedioTrajeto + R.
%Majorante
```

*Figura 16 - Predicado principal majorante*

Depois temos duas recursões distintas, uma recursão para se obter o maior tempo das parcelas “Trajeto”, “Cargas Adicionais” e “Tempos de descarga”, e outra para o tempo de carregamento de baterias.

Para a parcela “Trajeto”, desta vez devemos assumir a pior situação em que o camião vai mais pesado.

*“Contudo, dada a questão de as entregas serem diferentes em termos de massa poderá ocorrer uma situação em que embora o tempo com carga máxima seja menor seja transportada uma carga menor originando um tempo maior. Devemos então fazer os vários cálculos, vindo dos vários armazéns (lembrar que vindo de Matosinhos são transportados todos os 870 kg)”.*

Para isto, ao contrário do minorante que se ia obter o menor tempo de trajeto com o camião cheio e depois se fazia os cálculos a contar com a massa entregue, agora vamos à lista dos tempos, para cada um calcula-se o tempo de trajeto tendo já em conta a massa e só depois é que se compara o valor e coloca-se o maior para uma variável.

Ainda nesta recursão se adiciona também o tempo adicional.

```
%juntar todos os majorantes
recursaoMaj([],0):-!.
recursaoMaj([H|T],Maj):-recursaoMaj(T,Maj1),viagensParaIdMajorante(H,Z),
tempoAdicional(H,TempoAd),
Maj is Maj1+Z+TempoAd.
```

*Figura 13 - Recursão Majorante*

```
%Obter maior tempo da parcela trajeto
viagensParaIdMajorante(X,Z):-(findall(D,dadosCam_t_e_ta(_,_,X,D,_,_),R)),
findall(Tempo,entrega(_,_,_,_,_,Tempo),TemposEntrega),
somatorio(TemposEntrega,Q),
findall(Massa, entrega(_,_,Massa,_,_,_),Massas),
maiorTempoPercursoParaIdMaj(Massas,U),

listaTemposDepoisContas(U,R,Y),maiorTempoPercursoParaIdMaj(Y,W), Z is W+Q.
```

*Figura 14 - Predicado que retorna o tempo adicional + todas as descargas + maior tempo de trajeto pós contas*

Ainda neste predicado se soma todos os tempos de descarga no conjunto das entregas.

(Nota: MaiorTempoParaPercursoIdMajorante/2 é um predicado que somente retorna o maior valor de uma lista, usado em várias situações para evitar redundâncias.)

Para a última parcela, vamos a cada entrega, meter para uma lista todos os valores de energia necessário para se ir para aquele armazém, ver o maior, e guardar para a variável.
Soma-se as energias necessárias cada entrega para obter a energia necessária para entregar todas e depois aplicar a regra de três simples para se obter o tempo de carregamento em função da energia.

```
listaAtestar(X,L):-findall(D,dadosCam_t_e_ta(_,_,X,_,D,_), L).
%Obter maior energia necessária
tempoAtestar(X,TempoAd):-listaAtestar(X,List), maiorTempoPercursoParaIdMaj(List,TempoAd).

recursaoTempoCarregar([],0):-!.
recursaoTempoCarregar([H|T],Resultado):-recursaoTempoCarregar(T,Resultado1),tempoAtestar(H,Z), Resultado is Resultado1 + Z.
```

*Figura 15 - Tempo para carregar após soma de todas as maiores energias necessárias*

Somando para cada entrega todos os tempos de cada parcela, obtém-se o majorante de uma entrega, e depois somámos esses valores para obter o majorante total do conjunto de entregas.

```
?- majorante(M).
M = 1622.6000000000001.
```

*Figura 16 - Valor obtido de majorante*

- Heurísticas

O nosso grupo comprometeu-se a criar 3 heurísticas, nomeadamente entregar no armazém mais próximo, efetuar a ordem pelo peso das encomendas e por fim dando prioridade ao maior rácio (Massa da encomenda) / (Distância até ao respetivo armazém). Porém, não foi possível resolver o problema da última heurística mencionada.

```
heuristica(X,L):-
    findall(M,distancia(X,M,_),ListaAll),
    calculo_heuristica(X,ListaAll,L1),
    append(L1,[X],L).

calculo_heuristica(_,[],[]).
calculo_heuristica(X,ListaAux,[X|Trajeto]):-
    CaminhoInit(X,ListaAux,Z),
    member(Z,ListaAux),
    delete(ListaAux,Z,ListaAux1),
    calculo_heuristica(Z,ListaAux1,Trajeto).
          You, now • Uncommitted changes

CaminhoInit(Inicio,[C1|T],DestinoMaisPequeno):-Menor is 100000, caminhoPequeno(Inicio,[C1|T],_,Menor,DestinoMaisPequeno).

caminhoPequeno(_,[],DestinoMaisPequeno,_,DestinoMaisPequeno):-!.
caminhoPequeno(Inicio,[C1|T],_,MenorTemp,DestinoMaisPequenoTemp):-distancia(Inicio,C1,R), R<MenorTemp,!,
    caminhoPequeno(Inicio,T,C1,R,DestinoMaisPequenoTemp).

caminhoPequeno(Inicio,[_|T],DestinoMaisPequeno,MenorTemp,DestinoMaisPequenoTemp):-caminhoPequeno(Inicio,T,DestinoMaisPequeno,
MenorTemp,DestinoMaisPequenoTemp).
```

*Figura 16 – Heurística Menor Distância*

Na primeira heurística, que é possível ver implementada na figura acima, foram utilizados 4 predicados. O predicado heurística/2 guarda todos os possíveis destinos na ListaAll e envia para calculo_heuristica/3. A lista aí é percorrida através de uma chamada sucessiva do mesmo predicado, sendo sempre removido o destino encontrado a partir de CaminhoInit/5 antes da chamada. CaminhoInit/5 é responsável por iniciar a variável menor existindo assim desde o início um valor de comparação e envia, por fim, para caminhoPequeno/5, que face uma origem e uma lista de destinos possíveis encontra nesta mesma o armazém que será visitado de seguida, ou seja, o mais próximo dos que restam.

De seguida, com alguma inspiração na primeira, apresentamos a heurística que dá prioridade à massa das entregas, como é possível atentar na imagem abaixo:

```
%heuristicaMaiorMassa PI=PontoInicial

heuristicaMaiorMassa(PI,Data,L):-findall(Massa,entrega(_,Data,Massa,_,_,_),ListaAll),entrega(_,_,M,PI,_,_),delete(ListaAll,M,ListaAll1),
calculo_heuristica1(PI,ListaAll1,L1),append(L1,[PI],L).


calculo_heuristica1(_,[],[]).
calculo_heuristica1(PI,ListaAux,[PI|Trajeto]):- caminhoMaiorCarga(ListaAux,M),entrega(_,_,M,D,_,_), delete(ListaAux,M,ListaAux1),
calculo_heuristica1(D,ListaAux1,Trajeto).



caminhoMaiorCarga([],0):-!.
caminhoMaiorCarga([Massa|T],MaiorCarga):- caminhoMaiorCarga(T,MaiorCarga1),((Massa>MaiorCarga1,!,MaiorCarga is Massa);(MaiorCarga is MaiorCarga1)).
```

*Figura 17-Heurística Maior Massa*

Esta solução apresenta 3 predicados, sendo o primeiro(heuristicaMaiorMassa/3) onde são encontrados os pesos das entregas para um determinado dia(“Data”). Após isto, o calculo_heuristica1/3 trata de percorrer através de uma chamada sucessiva do mesmo predicado, sendo sempre removido o valor da massa encontrado a partir de caminhoMaiorCarga/3 antes da chamada. Finalmente este último predicado trata de encontrar a maior massa dentro da lista previamente recolhida.

```
%heuristicaMaiorRacioMassaDistancia PI=PontoInicial

%heuristicaMaiorRacio(PI,Data,L):-findall(Armazem,entrega(_,Data,_,Armazem,_,_),ListaAll),encontrarRacio(PI,ListaAll,ListaDestinos),
append(ListaDestinos,PI,L).

%encontrarRacio(,[],[]):-!.
%encontrarRacio(O,LA,[O|LD]):-caminhoMaiorRacio(O,LA,0,D),delete(LA,D,LA1),encontrarRacio(D,LA1,LD).



%caminhoMaiorRacio(_,[X],_,X):-!.
%caminhoMaiorRacio(PI,[X|LA],R,D):- distancia(PI,X,Dist),entrega(_,_,M,X,_,_),R is (M/Dist),D is X,caminhoMaiorRacio(PI,LA,R1,D1),
(R<R1,!,R is R1,D is D1).       You, now • Uncommitted changes
```

*Figura 18 – Heurística Maior Rácio Massa/Distância*

Finalmente, está acima presente a tentativa não conseguida de implementar a última heurística proposta. A ideia base ronda muito o que foi feito para as outras duas resoluções, com o acréscimo dos cálculos necessários para obter os rácios existentes entre massa de entrega e distância para o armazém onde esta será entregue. A solução apresentaria idealmente 3 predicados. Para este efeito, seria necessário encontrar os destinos das entregas para um determinado dia(“Data”) logo no primeiro predicado(heuristicaMaiorRacio/3). De seguida é percorrida a lista previamente preenchida sendo chamada de forma sucessiva o mesmo predicado encontrarRacio/3, sendo sempre removido o valor do destino encontrado a partir de caminhoMaiorRacio/4 antes da chamada. Por fim, seria encontrada a distância até certo destino e a massa da entrega destinada a esse mesmo, para ser definido depois o rácio de cada destino.

- Estudo da complexidade das heurísticas

| Nº de armazéns para Entrega | Solução | Tempo para Entregas Heurística da menor distância |
|---|---|---|
| 5 | [1,17,8,3,11,9,1] | 3.21e-5 s |
| 6 | [1,14,1,17,8,3,11,1] | 5.19e-5 s |
| 7 | [1,14,12,1,17,8,3,11,1] | 5.10e-5 s |
| 8 | [1,14,6,12,1,17,8,3,11,1] | 5.38e-5 s |
| 9 | [1,14,6,12,1,17,8,3,11,13,1] | 6.39e-5 s |
| 10 | [1,14,6,12,2,17,8,3,11,13,9,1] | 6.91e-5 s |
| 11 | [1,14,6,12,2,17,8,3,7,11,13,9,1] | 7.98e-5 s |
| 12 | [1,14,6,12,2,17,8,3,15,7,11,13,9,1] | 9.98e-5 s |
| 13 | [1,14,6,12,10,2,17,8,3,15,7,11,13,9,1] | 10,20e-5 s |
| 14 | [1,14,6,12,10,2,17,8,3,15,4,13,11,7,1,1] | 11,32e-5 s |
| 15 | [1,14,6,12,10,2,17,8,3,15,4,13,11,7,1,16,1] | 12,65e-5 s |

| Nº de armazéns para Entrega | Solução | Tempo para Entregas Heurística da maior massa |
|---|---|---|
| 5 | [1,17,11,9,8,3,1] | 2.10e-5 s |
| 6 | [1,17,11,14,9,8,3,1] | 2.23e-5 s |
| 7 | [1,17,11,14,12,9,8,3,1] | 2.41e-5 s |
| 8 | [1,17,11,14,6,12,9,8,3,1] | 2.60e-5 s |
| 9 | [1,13,17,11,14,6,12,9,8,3,1] | 2.78e-5 s |
| 10 | [1,13,2,17,11,14,6,12,9,8,3,1] | 2.98e-5 s |
| 11 | [1,13,2,17,11,7,14,6,12,9,8,3,1] | 3.60e-5 s |
| 12 | [1,13,2,17,11,7,14,15,6,12,9,8,3,1] | 4.32e-5 s |
| 13 | [1,13,2,10,17,11,7,14,15,6,12,9,8,3,1] | 4.00e-5 s |
| 14 | [1,13,2,10,17,11,7,14,15,4,6,12,9,8,3,1] | 5.22e-5 s |
| 15 | [1,13,2,10,16,17,11,7,14,15,4,6,12,9,8,3,1] | 5.48e-5 s |

- # Conclusão

  A partir das user stories desta cadeira que foram feitas recorrendo a matéria disponibilizada no Moodle e lecionada nas aulas, ficámos a perceber melhor como percorrer e permutar listas, descobrir o maior e menor valor, utilizar a recursividade nos predicados, partilhar variáveis entre predicados e implementar heurísticas.