Relatório de ALGAV

SPRINT 2

Turma 3DH _ Grupo 02
1201564 Jorge Ferreira
1201566 Rafael Leite
1201568 Rui Pina
1191008 Rodrigo Rodrigues

Data: 04/12/2022

Conteúdo

1.Base De Conhecimento

A base de conhecimento utilizada está dividida em três ficheiros (entregas_ex1.pl,entregas_ex2.pl,factos_camiao.pl,id_armazem.pl)

1.1 Entregas_ex1.pl

entrega(4439, 20221205, 200, "A01", 8, 10).
entrega(4438, 20221205, 150, "A09", 7, 9).
entrega(4445, 20221205, 100, "A03", 5, 7).
entrega(4443, 20221205, 120, "A08", 6, 8).
entrega(4449, 20221205, 300, "A11", 15, 20).
entrega(4441, 20221205, 301, "A11", 13, 20).

entrega(<idEntrega>,<data>,<massaEntrega>,<armazemEntrega>,<tempoColoc>,<tempoRet>)

O Ficheiro Entregas_ex1.pl contem os factos do tipo entrega que representam uma entrega individual, e que contêm dados importantes para realizar os cálculos necessários para a realização da sua distribuição de uma forma mais eficaz

1.2 Entregas_ex1.pl

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).

O ficheiro entregas_ex2.pl contem um número maior de entregas, de forma a realizar os predicados com um número maior de entregas.

1.3 Factos_camiao.pl

carateristicasCam(<nome_camiao>,<tara>,<capacidade_carga>,<carga_total_baterias>,<autonomia>,<t_recarr_bat_20a80>).

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

dadosCam_t_e_ta(<nome_camiao>,<cidade_origem>,<cidade_destino>,<tempo>,<energia>,<tempo_adicional>).

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).

…

O Ficheiro factos_camiao.pl contem factos do tipo caracteristicasCam e dadosCam_t_e_ta. Os factos caracteristicasCam representam as características de um camião e dados que serão usados no cálculo da duração das suas operações. Os factos dadosCam_t_e_ta representam dados específicos de um percurso de um armazém a outro para um camião específicos,dados que também são usados no calculo da melhor rota.

1.4 Id_armazem.pl

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).

O Ficheiro id_armazem.pl contem os factos do tipo idArmazem que representam a ligação entre os ids dos armazéns e os respetivos nomes.

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

Para implementar este requisito a subdivisão em pequenas tarefas era essencial, para isso, decidi subdividir este problema nas seguintes tarefas:

1. Encontrar as entregas numa determinada data e permutá-las de modo a ficar uma lista de listas;
2. Soma dos pesos das mercadorias a deixar pela rota para calcular o tempo gasto em cada etapa da rota;
3. Calcular a carga da bateria do camião a cada etapa para, posteriormente sabermos se temos carga das baterias suficiente para fazermos a próxima etapa da rota;
4. Tratar a informação recolhida até agora e atualizar a lista no predicado dinâmico custo_min/3, com as informações relativas ao melhor percurso, com sucessivas verificações de se o peso atual é inferior ao peso inferior anterior;

2.1. Encontrar as entregas numa determinada data e permutá-las de modo a ficar uma lista de listas

Primeira teremos como porta de entrada para este requisito este predicado. **melhorEntregaFindAll/1**.

```
melhorEntregaFindAll(Data):-retractall(custo_min(_, _ ,_)), assertz(custo_min(_, _,100000)),
retractall(data(_)), assertz(data(Data)),
encontrarTrajetos(Data, LP), melhorEntregaFindAllz(LP).
```

O utilizador irá passar como parâmetro a data da entrega, e o sistema encarregar-se-á de retornar no predicado dinâmico custo_min/3, a lista com o percurso de menor tempo.

Para isso, iremos pôr no terceiro parâmetro de custo_min um valor grande arbitrário, no nosso caso 100000, para realizar a primeira comparação entre o tempo do percurso atual e o tempo do percurso passado.

Além disso, iremos passar a data de pesquisa da entrega para o predicado dinâmico data/1, para que possamos usar com facilidade mais à frente.

```
armazensA(D,L):- findall(X,(entrega(_,D,_,X,_,_)),L).

encontrarTrajetos(Dia,LLTrajeto):-
armazensA(Dia,LA),findall(LTrajeto,permutation(LA,LTrajeto),LLTrajeto).
```

Por fim, passamos a data ao encontrarTrajetos/2 e este irá nos retornar uma lista de listas, com todas as permutações possíveis, utilizando o findall, para que possamos calcular o tempo de cada e, no fim, escolhermos a melhor.

```
melhorEntregaFindAllz([]):-!.
melhorEntregaFindAllz([H | L]):-
append([5 | H] , [5] , ListaComMatosinhos),
seq_custo_min(ListaComMatosinhos, LCcarregamentos,Custo),
melhorEntregaFindAllz(L).
```

Por último, neste passo, como temos uma lista de listas e iremos calcular o tempo de cada lista temos de percorrer a tal lista de listas, usando estre predicado. Para fazermos isto usamos o clássico método e mandar a Head da lista para o predicado seq_custo_min/3, fazendo os seus cálculos, e chamamos de forma recursiva melhorEntragaFindAllz/1 passando a Tail, até chegarmos à condição de paragem, que é a lista ficar vazia.

Nota: Como a lista de entregas na nossa base de conhecimento não contempla ter de começar e acabar em Matosinhos, requisito no caderno de encargos, tivemos de fazê-lo, adicionando-o com o append([5 | H], [5], ListaComMatosinhos), em que “5” representa o ID do armazém de Matosinhos.

2.2. Soma dos pesos das mercadorias a deixar pela rota para calcular o tempo gasto em cada etapa da rota

```
seq_custo_min(LC, LCcarregamentos,Custo):-(run(LC);true).

run(LC):-
calcula_custo(LC, Custo, LCcarregamentos),
atualiza(LC, Custo, LCcarregamentos), !,
fail.


atualiza(LCPerm, Custo, LCcarregamentos):-
custo_min(_,_,CustoMin),
((Custo<CustoMin,!,retract(custo_min(_,_,_)),assertz(custo_min(LCPerm, LCcarregamentos, Custo)),
write('Tempo='),write(Custo), write(' '),write(LCPerm), write(' com carregamentos em '),write(LCcarregamentos),nl);true).
% o write(Custo),nl so para ver a atualizacao do menor custo
```

Após chamarmos o seq_custo_min/3, iremos calcular o tempo (custo), da tal lista que era a Head da lista contendo todas as listas permutadas de entregas.

```
%c calcula_custo
calcula_custo(LC, Custo, LCcarregamentos):-
carateristicasCam(_, _, _, Carga_Total_Baterias, _, _),
custo(LC, Carga_Total_Baterias, Custo, LCcarregamentos).
```

Além da lista teremos de passar a carga total das baterias do camião (em KwH), neste momento um único. E enviaremos as 2 variáveis que irão guardar o custo e a lista de armazéns onde o camião terá de carrega, já que não tem bateria suficiente para fazer a próxima etapa da rota.

```
custo([_], _, 0, []).
custo([C1,C2|LC], Carga_Atual_Baterias, Custo, LCcarregamentos):-
dadosCam_t_e_ta(_,C1,C2,Tempo, Energia, TempoAdicional),
carateristicasCam(_, Tara, Capacidade_Carga, Carga_Total_Baterias, _, _),
soma_pesos([C1,C2|LC], LPesos, _),
acrescenta_tara(Tara, LPesos, [PesoComCarga|_]),
```

No predicado custo/4 iremos primeiramente buscar a informação necessária para o cálculo da carga das baterias e o tempo passado pelo camião em cada etapa, excluindo por enquanto o tempo de descarga e carga das mercadorias ou o tempo de carga da bateria de 20% a 80%, iremos fazer essa adição mais à frente.

```
%US2 a soma_pesos
soma_pesos([],[],0).

soma_pesos([5|LC],[PesoAc|LP],PesoAc):-
soma_pesos(LC, LP, PesoAc).

soma_pesos([Cidade|LC],[PesoAc|LP],PesoAc):-
data(Data),
soma_pesos(LC,LP,PesoAc1),entrega(_, Data, Peso, Cidade, _, _), PesoAc is Peso+PesoAc1.
```

Com essa informação iremos somar o peso da mercadoria deixada em cada etapa, fazendo-o de forma recursiva, adicionando a soma total da mercadoria para PesoAc e mete na lista em LP o total dos pesos da mercadoria que vão sendo subtraídas ao longo do percurso. Exemplo: Caso a lista de entregas tivesse como peso total 300kg e deixasse na primeira entrega 120kg e na segunda 150 a lista ficaria [300, 180, 30]. [1]

Como o armazém 5 (Matosinhos) é apenas de partida ao início do dia, não tem peso para deixar e de chegada ao fim do dia, é desprezível o peso já que não irá ter mais etapas, temos de passar todos os predicados que venham com o id "5" na Head, já que estes não têm peso associados na base de conhecimento.

```
%US2b acrescenta_tara
acrescenta_tara(Tara,[],[Tara]).
acrescenta_tara(Tara,[Peso|LP],[PesoTara|LPT]):-
acrescenta_tara(Tara,LP,LPT),
PesoTara is Peso+Tara.
```

Posteriormente adicionamos à tara e podermos mais à frente fazermos uma regra 3 simples para o cálculo do tempo, em relação à carga. Sendo a lógica semelhante a [1], adicionando os pesos já existentes em LP com a Tara.

2.3. Calcular a carga da bateria do camião a cada etapa para, posteriormente sabermos se temos carga das baterias suficiente para fazermos a próxima etapa da rota

```
custo([_], _, 0, []).
custo([C1,C2|LC], Carga_Atual_Baterias, Custo, LCcarregamentos):-
dadosCam_t_e_ta(_,C1,C2,Tempo, Energia, TempoAdicional),
carateristicasCam(_, Tara, Capacidade_Carga, Carga_Total_Baterias, _, _),
soma_pesos([C1,C2|LC], LPesos, _),
acrescenta_tara(Tara, LPesos, [PesoComCarga|_]),
calcular_carga_baterias(Energia, PesoComCarga, Tara, Capacidade_Carga, CargaNecessariaProximaViagem),
```

Com estas informações podemos agora calcular a carga da bateria que o camião irá necessitar a cada etapa. Precisamos, para isso de sabermos qual a Energia gasta na etapa, caso usemos o camião com a sua capacidade total; o peso com carga que tem atualmente nesta etapa, a tara, a capacidade total de carga do camião e o resultado, que irá ser armazenado na variável CargaNecessariaProximaViagem.

```
calcular_carga_baterias(CargaBateria, PesoComCarga, Tara, Capacidade_Carga, Resultado):-
Resultado is CargaBateria * PesoComCarga / (Tara + Capacidade_Carga).
```

Calculamos a Carga das baterias com uma regra 3 simples:

$$11800\ kg \leftrightarrow 42\ kWh$$
$$8500\ kg \leftrightarrow E$$
$$E = 42*8500/11800 = 30.25\ kWh$$ [2]

Usando este exemplo, podemos mostrar aquilo que foi dito anteriormente. Com a Capacidade Total do Camião (11800kg no exemplo) sabemos que o camião gasta um certo valor de kWh (42 kWh no exemplo), então com a carga total atual (soma das mercadorias + tara = 8500kg no exemplo) teremos E kWh

2.4. Tratar a informação recolhida até agora e atualizar a lista no predicado dinâmico custo_min/3, com as informações relativas ao melhor percurso, com sucessivas verificações de se o peso atual é inferior ao peso inferior anterior

```
((Carga_Atual_Baterias < CargaNecessariaProximaViagem, !, carateristicasCam(_, _, _, _, _, TCarga),
calcular_TCarga(Carga_Total_Baterias,Carga_Atual_Baterias, TCarga, TResultado),
A1 is Carga_Total_Baterias * 0.8 - CargaNecessariaProximaViagem, LCcarregamentos = [C1|LCcarregou]);
(A1 is Carga_Atual_Baterias  - CargaNecessariaProximaViagem, TResultado is 0, LCcarregamentos=LCcarregou)),
calcular_tempoCargaOuTempoColocarRetirar(TResultado, C2, TCargaResultado),
custo([C2|LC], A1, Custo1, LCcarregou),
Custo is Custo1 + (Tempo * PesoComCarga / (Tara + Capacidade_Carga)) + TCargaResultado + TempoAdicional.
```

Por fim iremos tratar toda esta informação e atualizar o custo_min/3 caso seja menor ao tempo

```
calcular_carga_baterias(CargaBateria, PesoComCarga, Tara, Capacidade_Carga, Resultado):-
Resultado is CargaBateria * PesoComCarga / (Tara + Capacidade_Carga).
```

anterior. Primeiramente, iremos ver se temos carga de bateria suficiente para fazer a proxima etapa da rota; se não tivermos iremos recarregar o camião até aos 80%, calculando o tempo de carregar com o predicado calcular_TCarga/4 e iremos adicionar a cidade onde estamos atualmente à lista de localidades onde teremos de carregar; se temos bateria suficiente apenas iremos subtrair a carga da próxima etapa à atual e iremos por TCarga como 0 já que não ouve carga. Com isto, iremos por fim ao calcular_tempoCargaOuTempoColocarRetirar/4.

Como temos informação do tempo a carregar dos 20% até aos 80%, temos de considerar não os 20%, mas a carga que temos atualmente, quer ela seja 13% ou 24%. Isto é uma regra 3 simples porque o tempo de carga é linear.

```
calcular_tempoCargaOuTempoColocarRetirar(TCarga, C2, TCargaResultado):-
data(Data),
entrega(_, Data, _, C2, TempoColocar, TempoRetirar),
TCarga < TempoColocar + TempoRetirar, !,
TCargaResultado is TempoColocar + TempoRetirar;
TCargaResultado is TCarga.
```

[3]

Aqui iremos ver qual dos tempos é maior: o tempo de carga ou o tempo de colocar e retirar o carregamento, pondo em TCargaResultado o tempo que for maior.

```
((Autonomia < CargaNecessariaProximaViagem, !, carateristicasCam(_, _, _, _, _, TCarga),
A1 is A - CargaNecessariaProximaViagem, LCcarregamentos = [C1|LCcarregou]);
(A1 is Autonomia * 0.8 - CargaNecessariaProximaViagem, TCarga is 0, LCcarregamentos=LCcarregou)),
calcular_tempoCarga(TCarga, C2, TCargaResultado),
custo([C2|LC], A1, Custo1, LCcarregou),
Custo is Custo1 + (Tempo * PesoComCarga / (Tara + Capacidade_Carga)) + TCargaResultado + TempoAdicional.
```

Por fim, iremos fazer isto recursivamente até não termos mais etapas para calcular na nossa rota e, termos o tempo final desta rota.

```
custo([_], _, 0, []).
```

Quando não houver mais pares de entregas iremos retornar aos passos passados e definir o Custo total da rota. Isto é feito com o valor do custo que temos naquele passo (0 no início) com a soma do tempo daquele passo que é constituído pela regra 3 simples semelhante a [2], só que considerando tempo em vez de energia, o TCargaResultado [3] e o TempoAdicional que é o tempo que o camião irá ter de carregar a meio da etapa, caso essa etapa passe os 80% da carga total, após a primeira saída do armazém de Matosinhos.

```
atualiza(LC, Custo, LCcarregamentos), !,
fail.


atualiza(LCPerm, Custo, LCcarregamentos):-
custo_min(_,_,CustoMin),
((Custo<CustoMin,!,retract(custo_min(_,_,_)),assertz(custo_min(LCPerm, LCcarregamentos, Custo)),
write('Tempo='),write(Custo), write(' '),write(LCPerm), write(' com carregamentos em '),write(LCcarregamentos),nl);true).
% o write(Custo),nl so para ver a atualizacao do menor custo
```

Finalmente, iremos atualizar o custo_min/3 caso o tempo atual for menor ao tempo mínimo que temos até então guardado no custo_min, caso o tempo atual for menor eliminamos o que está guardado em custo_min até então e pomos o novo custo mínimo. Além disso damos writes a esses valores para efeitos de demonstração. Caso o tempo atual seja maior que o valor até então não fazemos nada.

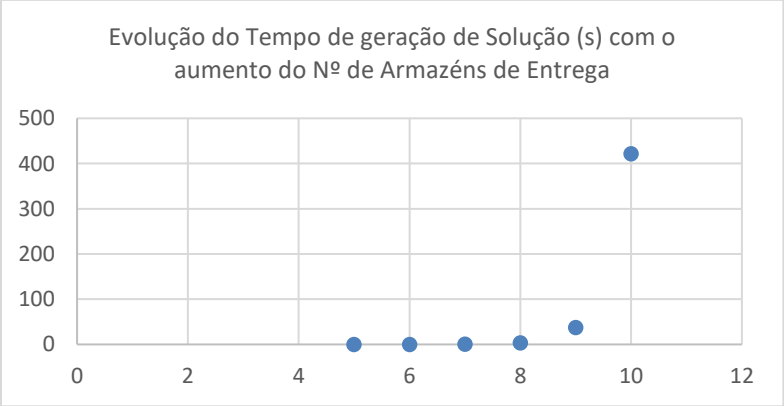
Com este procedimento todo ficamos com a rota que irá demorar menos tempo, tendo em atenção à carga de mercadorias do camião e à carga das baterias.

3. Aumentar a dimensão do problema (colocando mais armazéns a visitar) e verificar até que dimensão é viável proceder do modo adotado (com um gerador de todas as soluções) efetuando um estudo de complexidade do problema.

No estudo da complexidade vamos verificar como se comporta o método desenvolvido no ponto anterior, em função do aumento do Nº de Armazéns de Entrega. Para facilitar as nossas observações, desenvolvemos esta tabela com as colunas pertinentes para o estudo da complexidade.

Nº de Armazéns de Entrega	Nº de Soluções	Lista com sequência de Armazéns para as Entregas (1º e último são o Armazém Principal).	Tempo para fazer as Entregas (min)	Tempo de geração da Solução Tsol (s)
5	120	[5, 8, 1, 3, 11, 9, 5]	414.51250000000005	0.018249034881591797
6	720	[5, 17, 1, 3, 8, 11, 9, 5]	467.67457627118637	0.05595707893371582
7	5040	[5, 17, 14, 1, 3, 8, 11, 9, 5]	517.5991525423728	0.3709678649902344
8	40320	[5, 8, 3, 14, 1, 12, 17, 11, 9, 5]	578.4548728813559	3.3976800441741943
9	362880	[5, 17, 8, 14, 1, 12, 6, 3, 11, 9, 5]	603.9578389830509	37.575308084487915
10	3628800	[5, 17, 6, 14, 1, 12, 3, 8, 11, 13, 9, 5]	678.4701271186441	421.2687740325928

Como este método recorre do *"findall"*, obviamente que com o aumento do Nº de Armazéns de Entrega, tanto o Nº de Soluções como o Tempo de geração da Solução irão aumentar.



O Tempo de geração da Solução será superior em relação às heurísticas, pois neste método obtemos todos os caminhos possíveis em primeiro lugar *("findall")* e só depois procuramos pela solução que que demora menos Tempo para fazer as Entregas, o que não acontece nas heurísticas.

Este problema tem complexidade **O(n!),** em que o **n** corresponde ao Nº de Armazéns de Entrega. (O)

O Nº de Soluções está relacionado com o fatorial do Nº de Armazéns de Entrega.

- Nº de Soluções = (Nº de Armazéns de Entrega)!

O Tempo de geração da solução (a melhor solução) também depende do fatorial do Nº de Armazéns de Entrega, como podemos concluir mais facilmente com a observação do gráfico acima.

A partir de 10 Armazéns de Entrega é necessário aumentar a limite da stack para obtermos as soluções, sobrecarregando a memória da máquina que corre o predicado. Este método não é viável para listas de Entregas com vários Armazéns, daí a necessidade de desenvolvermos as heurísticas. Além disso, consideramos que não é exequível e prático um tempo de geração de solução superior a 60 segundos, pois este processo deve ser rápido e não impeditivo do normal funcionamento de uma empresa que utilize estes tipos de métodos.

4. Heurísticas para geração rápida de soluções

Para resolver o problema no nosso contexto, adequa-se muito o uso da heurística do problema do caixeiro-viajante. Isto porque, necessitamos de passar por todos os armazéns que têm as entregas, começando no início do dia em Matosinhos e acabando em Matosinhos.

O Problema do Caixeiro Viajante é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. E, portanto, foi aquele que escolhemos.

4.1. Uma que opte por ir para o armazém ao qual chegarmos em menor tempo no qual ainda não tenha sido feita a entrega

Começamos por preparar os dados a serem usados, primeiramente, ir buscar os armazéns onde terão entregas.

```
armazensComEntregas(D,L)
```

Inserir o armazém de Matosinhos, como partida e chegada, para ficar de acordo com o problema.

```
insereArmazemInicialFinal(N,[X|T],[X|L1])
```

E, calcular a massa total da rota.

```
massaTotalRotaI([X|T],S,D)
```

Com isto, podemos começar a tratar os dados, começando por obter a lista de armazéns com entregas do dia e obtemos id do armazém de Matosinhos.

```
maisProximo(Dia,RotaFinal) :- armazensComEntregas(Dia,Lista), idArmazem('Matosinhos',Id),
                write('Tempo Entre Armazens = '),maisProximoII(Lista,Rota,Id),insereArmazemInicialFinal(Id,Rota,RotaFinal),
                nl,write('Percurso = '),write(RotaFinal),write('\n'),
                calculosRota(RotaFinal,_,Dia),!.
```

Após isso vamos definir como armazém de partida o valor que vier no 3º argumento, neste caso inicial o Id de Matosinhos, e vamos comparar o tempo com os restantes armazéns de forma a receber o mais próximo no predicado maisProximoIII/4. Recebe o mais próximo, apaga-o da lista de armazéns a entregar, coloca esse como armazém de partida e repete o processo.

```
maisProximoII([],[ArmazemInicial],ArmazemInicial):-!.
maisProximoII(Lista,[ArmazemInicial|L1],ArmazemInicial):- maisProximoIII(ArmazemInicial,Lista,ArmazemProximo,T),
    write(T),write(' '),
                delete(Lista,ArmazemProximo,L2), maisProximoII(L2,L1,ArmazemProximo).
```

Com o armazém de partida, iremos comparar o seu tempo com o tempo das possibilidades e iremos retornar em ArmazemProximo o Armazém mais próximo, em relação ao que tínhamos atualmente.

```
maisProximoIII(_,[],_,0):- !.
maisProximoIII(ArmazemI,[H|T],ArmazemProximo, Tempo) :- dadosCam_t_e_ta(_,ArmazemI,H,T1,_,_), maisProximoIII(ArmazemI,T,A1,T2),
                    ((T2==0,!,Tempo is T1,ArmazemProximo = H);
                    (T1<T2,!,Tempo is T1, ArmazemProximo = H);
                    Tempo is T2,ArmazemProximo = A1).
```

4.2. Uma que opte por ir para o armazém onde e liberta a maior massa da entrega

Esta heurística será muito semelhante à anterior, mas iremos buscar pela maior massa e não pelo menor tempo. Com recurso aos predicados de obter os dados usados na heurística anterior, vamos agora para o armazém que tiver maior massa.

```
maiorMassa(Dia,RotaFinal) :- armazensComEntregas(Dia,Lista), idArmazem('Matosinhos',Id),
                write('Massa Entrega Entre Armazens = '),maiorMassaII(Lista,Rota,Id,Dia),insereArmazemInicialFinal(Id,Rota,RotaFinal),
                nl,write('Percurso = '),write(RotaFinal),write('\n'),
                calculosRota(RotaFinal,_,Dia),!.
```

Após isso vamos definir como armazém de partida o valor que vier no 3º argumento, neste caso inicial o Id de Matosinhos, e vamos comparar a massa com os restantes armazéns de forma a receber o que tiver maior no predicado maiorMassaIII/4. Recebe o que tiver mais massa, apaga-o da lista de armazéns a entregar, coloca esse como armazém de partida e repete o processo.

```
maiorMassaII([],[ArmazemInicial],ArmazemInicial,_):-!.
maiorMassaII(Lista,[ArmazemInicial|L1],ArmazemInicial,Dia):- maiorMassaIII(ArmazemInicial,Lista,ArmazemProximo,T,Dia),
    write(T),write(' '),
                delete(Lista,ArmazemProximo,L2), maiorMassaII(L2,L1,ArmazemProximo,Dia).
```

Com o armazém de partida, iremos comparar a sua massa com a massa das possibilidades e iremos retornar em ArmazemProximo o Armazém com maior massa, em relação ao que tínhamos atualmente.

```
maiorMassaIII(_,[],_,0,_):- !.
maiorMassaIII(ArmazemI,[H|T],ArmazemProximo, Tempo,Dia) :- entrega(_,Dia,T1,H,_,_), maiorMassaIII(ArmazemI,T,A1,T2,Dia),
                    ((T2==0,!,Tempo is T1,ArmazemProximo = H);
                    (T1>T2,!,Tempo is T1, ArmazemProximo = H);
                    Tempo is T2,ArmazemProximo = A1).
```

4.3. Uma que combine tempo com massa

Esta heurística será muito semelhante às outras duas, mas iremos calcular a relação entre a massa e o tempo.

```
combinadoTempoMassa(Dia,RotaFinal) :- armazensComEntregas(Dia,Lista), idArmazem('Matosinhos',Id),
                write('Custo Entre Armazens = '),combinadoTempoMassaII(Lista,Rota,Id),insereArmazemInicialFinal(Id,Rota,RotaFinal),
                nl,write('Percurso = '),write(RotaFinal),write('\n'),
                calculosRota(RotaFinal,_,Dia),!.
```

Vamos definir como armazém de partida o valor que vier no 3º argumento, neste caso inicial o Id de Matosinhos, e vamos comparar a relação do tempo com a massa com os restantes armazéns de forma

a receber o que tiver menor no predicado combinadoTempoMassaIII/4. Recebe o que tiver menor relação, apaga-o da lista de armazéns a entregar, coloca esse como armazém de partida e repete o processo.

```
combinadoTempoMassaII([],[ArmazemInicial],ArmazemInicial):-!.
combinadoTempoMassaII(Lista,[ArmazemInicial|L1],ArmazemInicial):- combinadoTempoMassaIII(ArmazemInicial,Lista,ArmazemProximo,T),
    write(T),write(' '),
                delete(Lista,ArmazemProximo,L2), combinadoTempoMassaII(L2,L1,ArmazemProximo).
```

Com o armazém de partida, iremos comparar o tempo com as possibilidades e iremos retornar em ArmazemProximo o Armazém que tiver maior valor na relação que fizemos, que no caso foi Massa * Tempo. Qualquer outra relação, que incorpora ambos os dados seria válida, por exemplo a divisão. Escolhemos a multiplicação por ser aquele que estava no exercício c de exame referido nos apoios.

```
combinadoTempoMassaIII(ArmazemI,[H|T],ArmazemProximo, Tempo) :- entrega(_,Dia,Massa,H,_,_), dadosCam_t_e_ta(_,ArmazemI,H,T1,_,_), combinadoTempoMassaIII(ArmazemI,T,A1,T2),
                ((T2==0,!,Tempo is T1 * Massa,ArmazemProximo = H);
                (T1<T2,!,Tempo is T1 * Massa, ArmazemProximo = H);
                Tempo is T2,ArmazemProximo = A1).
```

5. Análise da Qualidade das Heurísticas.

No estudo da complexidade vamos verificar como se comporta o método desenvolvido no ponto anterior, em função do aumento do Nº de Armazéns de Entrega. Para facilitar as nossas observações, desenvolvemos esta tabela com as colunas pertinentes para o estudo da complexidade.

Nº de Armazéns de Entrega	Solução ótima	Tempo para Entregas Solução ótima	Tempo para Entregas Heurística do menor tempo	Tempo para Entregas Heurística da maior massa	Tempo para Entregas Heurística combinada	Melhor solução pelas 3 heurísticas
5	[5, 8, 1, 3, 11, 9, 5]	0.018249034881591797	0.00950860977	0.00927591323	0.00943525267	[5,11,1,9,8,3,5]
6	[5, 17, 1, 3, 8, 11, 9, 5]	0.05595707893371582	0.00954079627	0.009291235118	0.00946744382	[5,17,11,1,9,8,3,5]
7	[5, 17, 14, 1, 3, 8, 11, 9, 5]	0.3709678649902344	0.00955057144	0.00931296345	0.00954899261	[5,17,11,14,1,9,8,3,5]
8	[5, 8, 3, 14, 1, 12, 17, 11, 9, 5]	3.3976800441741943	0.00935935974	0.00933256128	0.00958964245	[5,17,11,14,1,12,9,8,3,5]
9	[5, 17, 8, 14, 1, 12, 6, 3, 11, 9, 5]	37.575308084487915	0.01540112495	0.00934339144	0.00961253217	[5,17,11,14,6,1,12,9,8,3,5]
10	[5, 17, 6, 14, 1, 12, 3, 8, 11, 13, 9, 5]	421.2687740325928	0.00943684577	0.00934517388	0.00973826859	[5,13,17,11,14,6,1,12,9,8,3,5]
11	n/a	n/a	0.0095796585	0.00936221353	0.00949964548	[5,13,2,17,11,14,6,1,12,9,8,3,5]
12	n/a	n/a	0.00992321968	0.00936634739	0.00962142664	[5,13,2,17,11,7,14,6,1,12,9,8,3,5]
13	n/a	n/a	0.00947737693	0.00936898947	0.00931692123	[5,13,2,17,11,7,14,15,6,1,12,9,8,3,5]
14	n/a	n/a	0.00975036621	0.0093689983	0.00955963134	[5,13,2,10,17,11,7,14,15,6,1,12,9,8,3,5]
15	n/a	n/a	0.0096399784	0.00937104225	0.00946569442	[5,13,2,10,17,11,7,14,4,15,6,1,12,9,8,3,5]
16	n/a	n/a	0.00965285301	0.00939464569	0.00961661338	[5,13,2,10,16,17,11,7,14,4,15,6,1,12,9,8,3,5]

O Tempo de geração da Solução é inferior em relação às heurísticas, pois neste método não recorremos às permutações de todos os caminhos possíveis *("findall")* e só depois procuramos pela solução que que demora menos Tempo para fazer as Entregas.

Este problema tem complexidade **O (2^n * n²),** em que o **n** corresponde ao Nº de Armazéns de Entrega. Isto faz com que reduza de fatorial para exponencial.

6. Conclusões

Após a implementação das funcionalidades pedidas no âmbito da cadeira de ALGAV, podemos retirar as seguintes conclusões:

- Os métodos de encontrar caminhos com findall são menos eficientes do que métodos que não o usem.
- Com o método findall se tivermos muitas soluções obtemos um error de Stack Limit Exceeded, devido à sua complexidade de grandeza fatorial.
- Para obtermos resultados para bases de conhecimentos maiores que um certo patamar, 10 entregas do nosso caso, será necessário o uso de heurísticas, que, apesar de não nos retornar o melhor caminho, teremos sempre um resultado bom em relação ao ideal.