Support to Sprint B of Practical Work of ALGAV
Goods Delivery Planning with Electric Trucks

Curricular Unit: Advanced Algorithms

Programme: Computer Engineering (ISEP)

What is intended in Sprint B for the Planning module in ALGAV?

Sprint	**Module**	**Brief Description – User Story**
B	Planning	a. Receiving data on deliveries to be made by 1 truck and sections between warehouses: generating all possible trajectories through sequences of warehouses where deliveries must be made
B	Planning	b. evaluate these trajectories according to the time to complete all deliveries and return to the Matosinhos base warehouse and choose the solution that allows the return with the truck sooner
B	Planning	c. increase the dimension of the problem (putting more warehouses to be visited) and verify to what extent it is feasible to proceed in the adopted way (with a generator of all the solutions) by carrying out a study of the complexity of the problem
B	Planning	d. implement heuristics that can quickly generate a solution (not necessarily the best one) and evaluate the quality of these heuristics (e.g. deliver to the nearest warehouse; then deliver with greater mass; combine distance for delivery with mass delivered)
B	Planning	Only for groups with 5 or more elements: For a set of deliveries to be made, obtain a reduction and an increase in the time needed to make the respective deliveries

Note: the way SWI-Prolog communicates via HTTP, Json and server creation are not evaluated in ALGAV (it is evaluated just in LAPR5)

The problem we have in the work of ALGAV/LAPR5 is a classic TSP - “Traveling Salesman Problem”

- The traveling salesman leaves a city, he has to pass through all the other cities and in the end he returns to the city of departure
- Start/end city – Matosinhos
- Cities you have to go through – all where goods are delivered
- There are direct connections between any pair of cities
- This is our problem
- The problem has n! complexity (where n is the number of cities that the traveling salesman must visit, excluding the departure/arrival city of Matosinhos)

The problem we have in the work of ALGAV/LAPR5 is a classic TSP - “Traveling Salesman Problem”

- For example, the truck leaves Matosinhos and has to pass through Maia, Porto and Espinho, returning at the end to Matosinhos, the solutions are 6 (3!):
  - [matosinhos, maia, porto, espinho, matosinhos]
  - [matosinhos, maia, espinho, porto, matosinhos]
  - [matosinhos, porto, maia, espinho, matosinhos]
  - [matosinhos, porto, espinho, maia, matosinhos]
  - [matosinhos, espinho, maia, porto, matosinhos]
  - [matosinhos, espinho, porto, maia, matosinhos]

- If we omit Matosinhos at the beginning and end we get:
  - [maia, porto, espinho]
  - [maia, espinho, porto]
  - [porto, maia, espinho]
  - [porto, espinho, maia]
  - [espinho, maia, porto]
  - [espinho, porto, maia]

The problem we have in the work of ALGAV/LAPR5 is a classic TSP - “Traveling Salesman Problem”

- Let's remember the last problem of the list of problems n. 2 (Lists) in which we were asked to create a permutacao/2 predicate that receiving a list of elements in the first argument generates a permutation of these elements. If all solutions are asked for, this predicate will give all possible permutations

```
?-permutacao([maia, porto, espinho],LP).
    LP=[maia, porto, espinho];
    LP=[maia, espinho, porto];
    LP=[porto, maia, espinho];
    LP=[porto, espinho, maia];
    LP=[espinho, maia, porto];
    LP=[espinho, porto, maia]
```

And if we want to generate all solutions:

```
?-findall(LP, permutacao([maia, porto, espinho],LP), LLP).
LLP=[[maia, porto, espinho], [maia, espinho, porto], [porto, maia, espinho], [porto, espinho,
maia], [espinho, maia, porto], [espinho, porto, maia]]
```

SWI-Prolog has already a pre-defined permutation/2 predicate doing the generation of a permutation of a list

Generating all solutions with findall … Is that a good idea?

```
?- findall(LP, permutation([a,b,c,d,e],LP),LLP).
LLP = [[a, b, c, d, e], [a, b, c, e, d], [a, b, d, c, e], [a, b, d, e, c], [a, b, e, c|...], [a, b, e|...], [a, c|...], [a|...], [...|...]|...].

?- findall(LP, permutation([a,b,c,d,e,f],LP),LLP).
LLP = [[a, b, c, d, e, f], [a, b, c, d, f, e], [a, b, c, e, d, f], [a, b, c, e, f|...], [a, b, c, f|...], [a, b, c|...], [a, b|...], [a|...], [...|...]|...].

?- findall(LP, permutation([a,b,c,d,e,f,g],LP),LLP).
LLP = [[a, b, c, d, e, f, g], [a, b, c, d, e, g, f], [a, b, c, d, f, e|...], [a, b, c, d, f|...], [a, b, c, d|...], [a, b, c|...], [a, b|...], [a|...], [...|...]|...].

?- findall(LP, permutation([a,b,c,d,e,f,g,h],LP),LLP).
LLP = [[a, b, c, d, e, f, g, h], [a, b, c, d, e, f, h|...], [a, b, c, d, e, g|...], [a, b, c, d, e|...], [a, b, c, d|...], [a, b, c|...], [a, b|...], [a|...],
[...|...]|...].

?- findall(LP, permutation([a,b,c,d,e,f,g,h,i],LP),LLP).
LLP = [[a, b, c, d, e, f, g, h|...], [a, b, c, d, e, f, g|...], [a, b, c, d, e, f|...], [a, b, c, d, e|...], [a, b, c, d|...], [a, b, c|...], [a, b|...], [a|...],
[...|...]|...].

?- findall(LP, permutation([a,b,c,d,e,f,g,h,i,j],LP),LLP).
ERROR: Stack limit (1.0Gb) exceeded
```

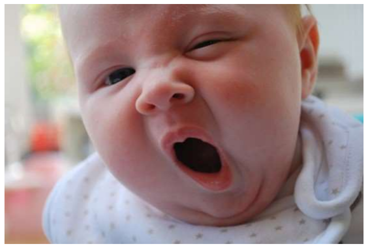
Generating all solutions with findall … Is that a good idea?

But if the stack was not enough we can resize it

```
?- set_prolog_flag(stack_limit, 10 000 000 000).
true.
```

```
?- findall(LP, permutation([a,b,c,d,e,f,g,h,i,j],LP),LLP).
LLP = [[a, b, c, d, e, f, g, h|...], [a, b, c, d, e, f, g|...], [a, b, c, d, e, f|...], [a, b, c, d, e|...], [a, b,
c, d|...], [a, b, c|...], [a, b|...], [a|...], [...|...]|...].
```

```
?- findall(LP, permutation([a,b,c,d,e,f,g,h,i,j,k],LP),LLP).
```



ERROR: Stack limit (9.3Gb) exceeded

Increase the size of the Stack allows larger dimension lists, but we now have the generation time problem

Let's go sooner or later reaching the limit where the stack size is no longer sufficient

This will force us, from a given dimension, to opt for heuristics that do not guarantee the optimal solution

If a problem is difficult then break it down into phases that make the solution simpler!

The problem at hand is difficult and complex (affected by the combinatorial explosion).

As for “difficult” we will divide it into phases, and we will go step by step considering new phases.

The problem involves the distribution of goods with an electric truck, which requires careful management to ensure that the battery charge does not fall below a certain threshold (20%).

The proposal is, in phase 1, that we only deal with the problem of distributing the goods, admitting that the battery charge did not have a maximum limit of 80 kWh, but a much higher limit that guaranteed us that all deliveries could be made without the need to charge the batteries midway.

Once the problem has been solved for phase 1, we will move on to phase 2 where we will see the limitations that real batteries impose and that have consequences in the total solution time.

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

Consider a problem in which a factory wants to optimize the cost of deliveries it has to make to its customers by aggregating deliveries from customers from the same city into a given load and using a truck to deliver in those cities. The cost will be related to the distances traveled and the weight moved, including the tare (weight of the truck itself) and the weight of loads for cities.

Loads for cities are represented by facts of the type *carga(City,Load_Weight)* which indicates that for a given *City* we have to deliver orders with total weight *Load_Weight*, where the weight is in tons. For example, we could have the following loads in the knowledge base:

```
carga(a,0.5).
carga(b,1.8).
carga(c,1).
carga(d,2.5).
carga(e,1.2).
carga(f,2).
```

"carga" means load in portuguese

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

a)[20%] Write the predicate *soma_pesos(LCities,LWeights,TotalWeight)* assuming that the cities will be travelled in the order in which they appear in the *LCities* instantiated list and as result, put in *LWeight* the total value of the load with which the truck arrives in the city (that is, the load destined for the city and the sum of loads for the following cities on the route) and in *TotalWeight* will be the sum of weights of all loads, that is, the total load with which the truck leaves the factory to make deliveries. For example, the predicate call could be as follows:

```
?- soma_pesos([a,b,c,d,e,f],LPesos,PesoTotal).
LPesos = [9.0, 8.5, 6.7, 5.7, 3.2, 2],
PesoTotal = 9.0.
```

```
carga(a,0.5).
carga(b,1.8).
carga(c,1).
carga(d,2.5).
carga(e,1.2).
carga(f,2).
```

"soma_pesos" means "sum_weights" in portuguese
PesoTotal means TotalWeight

```
%a soma_pesos

soma_pesos([],[],0).
soma_pesos([Cidade|LC],[PesoAc|LP],PesoAc):-
   soma_pesos(LC,LP,PesoAc1),carga(Cidade,Peso),PesoAc is Peso+PesoAc1.
```

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

b)[20%] Write the predicate *acrescenta_tara(Tare,LWeights,LWeightsTare)* that for a given *Tare* (weight of the truck without load) expressed in tons and a list of weights *LWeights*, as the one obtained in a), returns a list *LWeightsTare* that adds the truck's *Tare* to the *LWeights* elements and puts more an element at the end of the list equal to the *Tare* of the truck (it will be the weight of the empty truck to return to the starting city where the factory is after delivering all the loads). For example, the predicate call could be as follows:

?- acrescenta_tara(6,[9.0, 8.5, 6.7, 5.7, 3.2, 2],LPesosTara).

LPesosTara = [15.0, 14.5, 12.7, 11.7, 9.2, 8, 6] .

"acrescenta_tara" means "add_TruckWeight" in portuguese
LPesoTara means LTruckWeight

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

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

c)[30%] Assume now that the knowledge base involves facts *dist(City1,City2,Distance)* about distances *Distance* between cities *City1* and *City2* in km (assume that the fact represents a bidirectional relationship and that there is always a fact for any pair of cities). Assume that the factory city is represented by a fact *cidade_fabrica(City)*. Assume that the tare weight of the truck is represented by the fact *tara(WeightTruck)*. Write the predicate *calcula_custo(LCities,Cost)* which calculates the cost as the sum of the products of the distances covered by the moving weight (of the truck plus transported loads). As an example, let's consider, in addition to the *load/2* facts already mentioned above, that we have the following facts in the knowledge:

```
tara(6).
cidade_fabrica(i).
dist(i,a,40).
dist(i,b,40).
dist(i,c,5).
dist(a,b,50).
dist(a,c,30).
dist(b,c,25).
%...mais distancias entre cidades
```

"tara" means "truck weight" in portuguese
"cidade_fabrica" means "factory_city" in portuguese

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

c) (continuation) …

For exemple, the cal to the predicate calcula_custo/2 would be the following:

```
?- calcula_custo([a,b,c],Custo).
Custo = 1017.0 .
```

Where the cost would be calculated for the path [i,a,b,c,i] to which the distances [40,50,25,5] are associated, which would be multiplied by the gross weights (tare plus load) [9.3,8.8, 7.6] (9.3=6 tonnes of the truck plus 3.3 tonnes of the 3 loads for a,b, and c; 8.8=6 tonnes of the truck plus 2.8 tonnes of the loads for b and c; 7=6 tonnes of the truck plus 1 ton for c; and 6 tons is the weight of the truck to return to i). So 1017 results from the calculation 40*9.3 + 50*8.8 + 25*7 + 5*6

"calcula_custo" means "calculate_cost" in portuguese
"Custo" means "Cost" in portuguese

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

```
%c calcula_custo
calcula_custo(LC,Custo):-
   soma_pesos(LC,LP,_),
   tara(Tara),
   acrescenta_tara(Tara,LP,LPT),
   cidade_fabrica(Cin),
   append([Cin|LC],[Cin],LCcompleto),
   custo(LCcompleto,LPT,Custo).


custo([_],[],0).
custo([C1,C2|LC],[PT|LPT],Custo):-
   custo([C2|LC],LPT,Custo1),
   (dist(C1,C2,Dist);dist(C2,C1,Dist)),
   Custo is Custo1+Dist*PT.
```

```
tara(6).

cidade_fabrica(i).

dist(i,a,40).
dist(i,b,40).
dist(i,c,5).
dist(a,b,50).
dist(a,c,30).
dist(b,c,25).
```

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

d)[30%] Write a predicate *seq_custo_min(LC,Cost)* predicate that determines the sequence of deliveries of all loads that leads to the lowest cost according to the cost calculation defined in c), returning in *LC* the sequence of cities in the delivery path and in *Cost* the respective cost. The cities to be delivered are those that are in the carga/2 facts.

Suggestions:

-if you get a list with all the cities, you can generate all the sequences of cities through the permutation of elements of the list and for this purpose you can use the predicate already existing in the SWI Prolog *permutation(ListCities,ListPermutedCities)*

-store in a dynamic fact *custo_min(PermutedCitiesList,Cost)* the lowest *Cost* of the best permuted list of cities *PermutedCitiesList* while the costs of the different city permutations are being calculated, in the end the fact will have the best solution

For example, a call to the predicate would be the following:

```
?- seq_custo_min(LC,Custo).

LC = [c, d, a, b, e, f],

Custo = 2664.5.
```

To help with phase 1 (only the distribution part of the goods) we are going to use a problem of the assessment exam of last year

Problem of the Exam 07/02/2022

```
%d seq_custo_min
seq_custo_min(LC,Custo):-(run;true),custo_min(LC,Custo).

run:- retractall(custo_min(_,_)),  assertz(custo_min(_,100000)),
   findall(Cidade,carga(Cidade,_),LC),
   permutation(LC,LCPerm),
   calcula_custo(LCPerm,Custo),
   atualiza(LCPerm,Custo),
   fail.

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

It is not necessary to save all solutions,we can, as we generate them, go keeping the best solution so far,then using a "retract"-"assert", which removes the previous best and saves the new best solution.
The fail is set to force you to go back and generate a new path and ensure that all solutions will be generated)

How is our problem different from that of the exam?

The Knowledge Base envolves different facts:

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

dadosCam_t_e_ta(eTruck01,1,2,122,42,0).

%...

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

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

%...

%idArmazem(<local>,<codigo>)

idArmazem('Arouca',1).

%...

%entrega(<idEntrega>,<data>,<massaEntrefa>,<armazemEntrega>,<tempoColoc>,<tempoRet>).

entrega(4439, 20221205, 200, 1, 8, 10).

%...
```

"dadosCam" means "dataTruck" in portuguese
"carateristicas" means "features" in portuguese
"Armazem" means Warehouse
"entrega" means delivery
"tempo" means time
"cidade" means city

How is our problem different from that of the exam?

The Knowledge Base envolves different facts (cont.):

```
%cidadeArmazem(<codigo>).

cidadeArmazem(5).
```

Distances are no longer important and what becomes important is time

Work proposal for the 1st week of the ALGAV Sprint B:

-Adapt the solution of the 7/2/2022 exam problem to the problem stated in the ALGAV work of 2022/2023

-Just consider phase 1, for now we will only think about the distribution of goods and assume that the truck battery would not need to be recharged

And after having the problem of the distribution of goods working well, we will think about the issue of charging the batteries of the electric truck