ALGAV - Project Sprint B Report

- Ibon - 1220417
- Jakub - 1220419
- João Silva - 1200616
- Pedro Marques - 1191606
- Diogo Marques - 1190516

USER STORIES

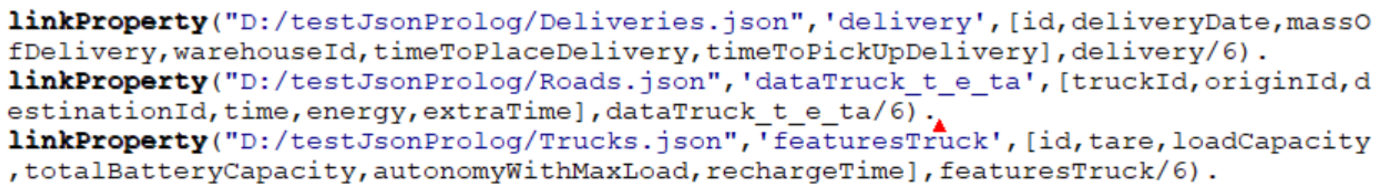
- US 3.2.3 - 1.a) The receiving data on deliveries to be made by 1 truck and on sections between warehouses: generating all possible trajectories through sequences of warehouses where deliveries must be made.
- US 3.2.3 - 1.b) Evaluate these trajectories according to the time to complete all deliveries and return to the Matosinhos base warehouse and choose the solution that allows the truck to return sooner.
- US 3.2.3 - 1.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.

- US 3.2.3 - 1.d) Implement heuristics that can quickly generate a solution (not necessarily the best one) and evaluate the quality of these heuristics (for example, deliver to the nearest warehouse; then deliver with greater mass; combine distance for delivery with mass delivered).
- US 3.2.3 - 1.e) For a set of deliveries to be made, obtain a minor and a maximum of the time required to make the respective deliveries.

Domain knowledge representation

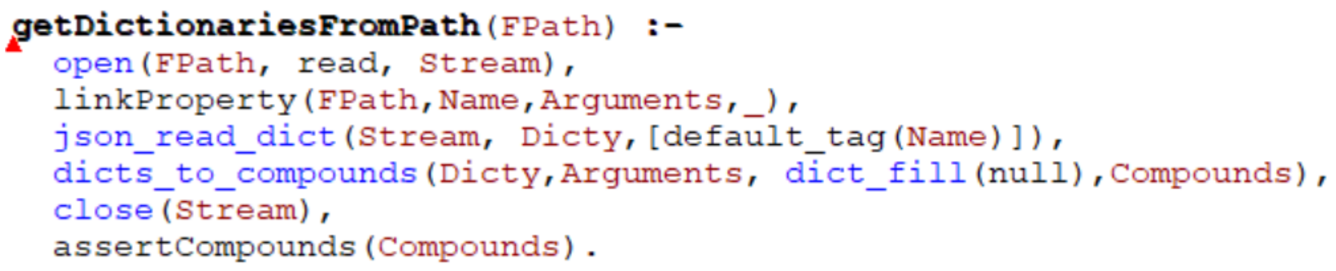
Knowledge base for this project is adjusted to be taken from the database through REST Api. For testing the module and representing heuristics and review we used parser from json response text format that will be delivered as a response from Api and mocked facts in PROLOG format.

For testing part we take data from JSONs specified in the following facts:



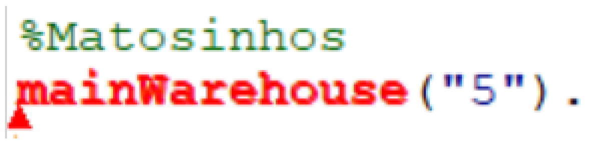
```
linkProperty("D:/testJsonProlog/Deliveries.json",'delivery',[id,deliveryDate,massO
fDelivery,warehouseId,timeToPlaceDelivery,timeToPickUpDelivery],delivery/6).
linkProperty("D:/testJsonProlog/Roads.json",'dataTruck_t_e_ta',[truckId,originId,d
estinationId,time,energy,extraTime],dataTruck_t_e_ta/6).
linkProperty("D:/testJsonProlog/Trucks.json",'featuresTruck',[id,tare,loadCapacity
,totalBatteryCapacity,autonomyWithMaxLoad,rechargeTime],featuresTruck/6).
```

It is read from files with use of streams and then written in the memory with the following code, where assertCompounds is a loop with assertz for all the facts:



```
getDictionariesFromPath(FPath) :-
   open(FPath, read, Stream),
   linkProperty(FPath,Name,Arguments,_),
   json_read_dict(Stream, Dicty,[default_tag(Name)]),
   dicts_to_compounds(Dicty,Arguments, dict_fill(null),Compounds),
   close(Stream),
   assertCompounds(Compounds).
```

In a knowledge base that for now is not imported from the database (will be in a future with use of e.g. flag isMainWarehouse) we have a fact with main warehouse, which will be begining and end of all trips:



```
%Matosinhos
mainWarehouse("5").
```

Truck

Following example is based on the assumption that we do not use more than one truck (which will be upgraded to many trucks in the next sprint):

Example of the fact regarding truck:

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

- Truck identification
- Weight of the truck
- Weight capacity of the truck
- Total batteries capacity
- Distance on single charge
- Time necessary to charge the truck between 20% and 80%

For making code cleaner we use helping functions to get the details about truck that we need to obtain at the moment:

- Weight of the truck
- Capacity of the truck
- Total possible weight of the truck (Weight + Capacity)
- Total energy that can be stored in batteries (kWh)
- Time to charge batteries from 20% to 80%

```
truckWeight(TW):-featuresTruck(_,TW,_,_,_,_).
capacityWeight(CW):-featuresTruck(_,_,CW,_,_,_).

totalTruckWeight(TTW):-truckWeight(TW),capacityWeight(CW),TTW is TW+CW.

totalBatteries(TB):- featuresTruck(_,_,_,TB,_,_).
timeToRechargeBatteries(TR):-featuresTruck(_,_,_,_,_,TR).
```

Data Truck

Example of the facts regarding data truck:

```
dataTruck_t_e_ta(eTruck01,5,1,141,51,24).
dataTruck_t_e_ta(eTruck01,1,9,185,74,53).
dataTruck_t_e_ta(eTruck01,9,3,86,35,0).
dataTruck_t_e_ta(eTruck01,3,8,38,8,0).
dataTruck_t_e_ta(eTruck01,8,11,53,22,0).
dataTruck_t_e_ta(eTruck01,11,5,55,25,0).
```

- Identification of the truck
- Id of the warehouse on the beginning of the trip
- Id of the warehouse on the end of the trip
- Time necessary to accomplish the trip with full load
- Energy necessary to accomplish the trip with full load

Here as a helpers we use following code to take info about:

- Time between one and the other warehouse
- Added time for charging the batteries on the way
- Energy consumption on this distance

```
timeWarehouses(C1,C2,T):-dataTruck_t_e_ta(_,C1,C2,T,_,_).

addTime(C1,C2,AT):-dataTruck_t_e_ta(_,C1,C2,_,_,AT).

energyW(C1,C2,EW):-dataTruck_t_e_ta(_,C1,C2,_,EW,_).
```

Delivery

Example of the facts regarding deliveries:

```
delivery(4439, 20221205, 200, 1, 8, 10).
delivery(4438, 20221205, 150, 9, 7, 9).
delivery(4445, 20221205, 100, 3, 5, 7).
delivery(4443, 20221205, 120, 8, 6, 8).
delivery(4449, 20221205, 300, 11, 15, 20).
```

- Identification of delivery
- Date of the delivery
- Weight that need to be delivered to warehouse

- Destination warehouse for the delivery
- Placement time for the delivery
- Pick up time for the delivery

Here as a helpers we use following code to take info about:

- Load weight that need to be delivered to the specific warehouse
- Total time necessary to do the pickup and unload the truck

```
loadWeight(WarehouseID,Weight):-delivery(_,_,Weight,WarehouseID,_,_).

pickupT(C,PT):-delivery(_,_,_,C,_,PT).
```

Optimal solution for Deliveries Planning with an electrical truck

We use following code to recursively calculate the sum of Weight needed to be transported during the road to each city:

```
sumOfWeight([],[],0).
sumOfWeight([City|LC],[WeightAc|LW],WeightAc):-
    sumOfWeight(LC,LW,WeightAc1),loadWeight(City,Weight),WeightAc is Weight+Weight
Ac1.
```

Following code add weight of the truck to the previous calculations:

```
addTruckWeight(TruckW,[],[TruckW]).
addTruckWeight(TruckW,[Weight|LW],[TotalTW|LTW]):-addTruckWeight(TruckW,LW,LTW), T
otalTW is Weight+TruckW.
```

Following code calculate necessary time for driving, pick up and charging the truck combined:

```
calculateTimes(LC,TotalTime):-
    sumOfWeight(LC,LW,_),
    truckWeight(TruckW),
    addTruckWeight(TruckW,LW,LTW),
    mainWarehouse(MainCW),
    append([MainCW|LC],[MainCW],TotalLC),
    totalBatteries(Batteries),
    calculateTimesInternal(TotalLC,LTW,_,TotalWayTime),
    calcEnergy(TotalLC,LTW,Batteries,TChargingTime),
    nl,nl,
    write("Total pickup and charging time: "),write(TChargingTime),nl,
    write("Total road trips time: "),write(TotalWayTime),nl,nl,
    TotalTime is TChargingTime+TotalWayTime,
    write("Total time: "),write(TotalTime),nl,nl.
```

It makes use of the helpers functions presented above to get weight of the truck, main warehouse city (which is added on the beginning and end of the list of cities), total battery capacity.

Then inside the calculateTimeInternal the time necessary for all the time necessary for transport between cities is calculated:

```
calculateTimesInternal([_],_,[],0).
calculateTimesInternal([C1,C2|LC],[TW|LTW],[Time|LTimes],TotalTime):-
    calculateTimesInternal([C2|LC],LTW,LTimes,TotalTime1),
    %change to dataTruck_t_e_ta here
    timeWarehouses(C1,C2,FullWeightTime),
    totalTruckWeight(TTW),
    Multiplier is TW/TTW,
    Time is FullWeightTime*Multiplier,
    TotalTime is TotalTime1+Time.
```

Inside this function we receive data from helpers about time needed for transport from first to second warehouse and total weight of the truck. We use a multiplier calculated based on the ratio between loaded weight and total capacity. Following multiplier gives us the part of total value that we should consider for calculations.

For calculation of time necessary for charging batteries we calculate based on the following code:

```
calcEnergy(LC,LTW,TotalBatteries,TimeSum):-
    calculateEnergyInternal(LC,LTW,TotalBatteries,TotalBatteries,TimeSum).

calculateEnergyInternal([_],_,_,_,0).
calculateEnergyInternal([C1,C2|LC],[TW|LTW],PreviousE,TotalBatteries,TimeSum):-
    totalTruckWeight(TTW),
    Multiplier is TW/TTW,
    nl,write("Multiplication: "),write(Multiplier),nl,
    limits(TotalBatteries,LimitBottom,LimitTop),
    energyW(C1,C2,FullWeightEnergy),
    EnergyConsumption is FullWeightEnergy*Multiplier,
    NecessaryE is LimitBottom + EnergyConsumption,
    addTime(C1,C2,AdditionalT),
    write("Additional time is "),write(AdditionalT),nl,

    levelToCharge(LC,LimitTop,NecessaryE,LoadingLevel),
    (pickupT(C1,PickupT);PickupT is 0),
    ((PreviousE<NecessaryE,!,calculateChargeTime(PreviousE,LoadingLevel,LimitTop,L
imitBottom,ChargeT,NewE),(
   (AdditionalT>0,!,AfterE is LimitBottom)
    ;AfterE is NewE-EnergyConsumption))
  ;(AfterE is  PreviousE-EnergyConsumption,ChargeT is 0)),
    ((ChargeT<PickupT,!,CPT is PickupT);(CPT is ChargeT)),
    write("ChargePickupTime is "),write(CPT),nl,
    calculateEnergyInternal([C2|LC],LTW,AfterE,TotalBatteries,TimeSum1),
    TimeSum is TimeSum1+CPT+AdditionalT.
```

We get from helper functions info about total weight of truck, bottom and top limits of battery level during transportation (20% and 80% respectively), energy needed for transport with full weight, added time for charging between warehouses and pickup time.

Our logic is based on assumption that during charging a truck we can also do the pick up. We take into calculations the time that takes longer to accomplish. Before each trip we check recursively if it is possible to accomplish the road without recharging batteries before departure. We also take into account level to which we need to charge, if it is beginning of the trip we have more than a limit, and on the last trip it is necessary to charge truck to the extent that allow us to come back to warehouse with just 20% of batteries.

```
levelToCharge([],_,NecessaryE,NecessaryE).
levelToCharge(_,LimitTop,_,LimitTop).

calculateChargeTime(PreviousE,LoadingLevel,LimitTop,LimitBottom,ChargeT,NewE):-
    ((PreviousE>=LoadingLevel,!,NewE is PreviousE,ChargeT is 0);(
        timeToRechargeBatteries(RechargeT),
        NewE is LoadingLevel,
        ChargeT is  (LoadingLevel-PreviousE)*RechargeT/(LimitTop-LimitBottom)
     )).
```

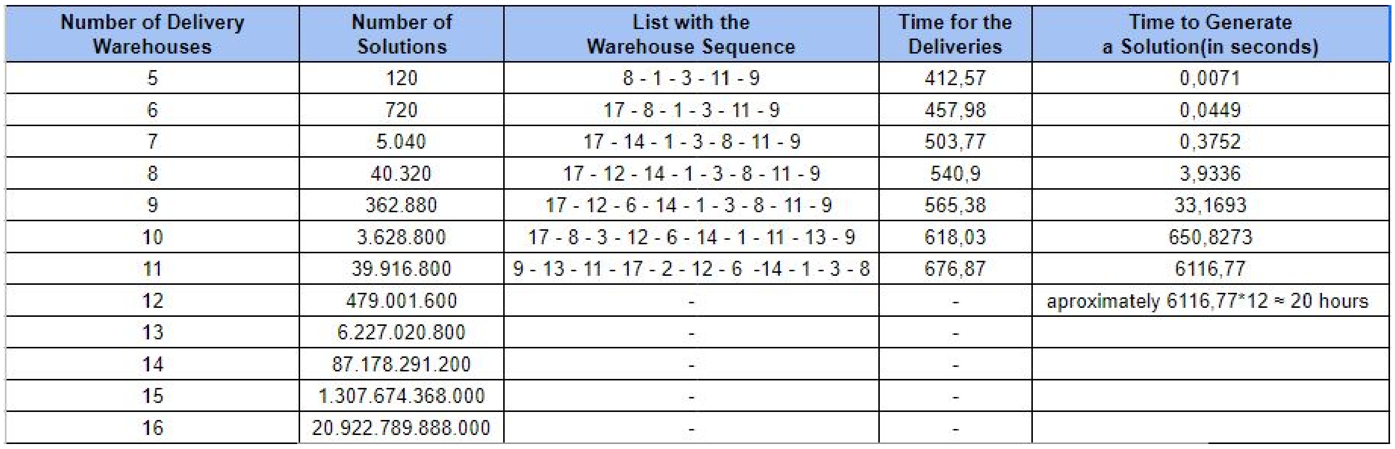
For calculating the best permutation of small list of given deliveries destination we use following code:

```
lowestTimeSequence(LC,Time):-(runLCS;true),lowestTime(LC,Time).

runLCS:- retractall(lowestTime(_,_)), assertz(lowestTime(_,10000000)),
    findall(City, loadWeight(City,_),LC),
    permutation(LC,LCPerm),
    calculateTimes(LCPerm,_,Time),
    updateLowest(LCPerm,Time),
    fail.


updateLowest(LCPerm,Time):-
    lowestTime(_,TimeMin),
    ((Time<TimeMin,!,retract(lowestTime(_,_)),assertz(lowestTime(LCPerm,Time)),
      write(Time),nl);true).
```

When it finds the permutation with smaller time necessary to accomplish the trip it save it into the memory and the previous lowest value is droped.

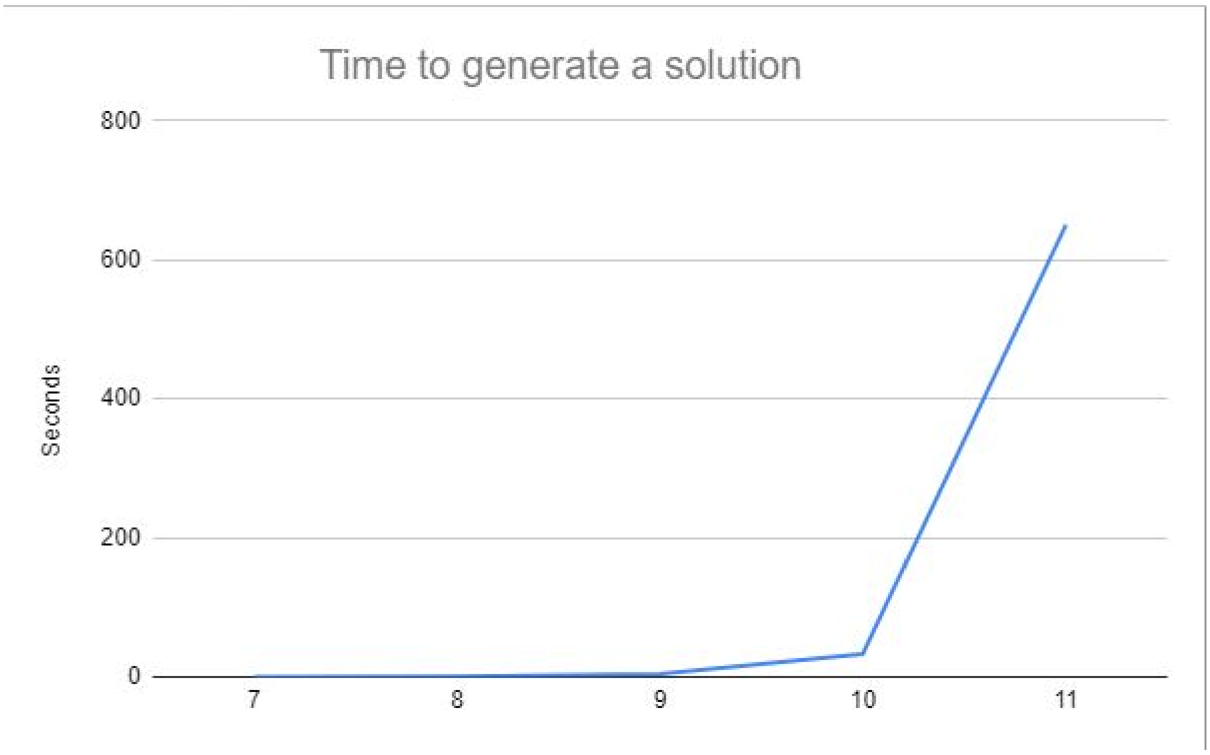
Study of the problem complexity as well as the viability of finding the optimal solution by generating all the solutions

The problem has been scaled from 5 to 17 warehouses, using the table of ex deliveries 2, given in the Algav Moodle page. With with information it has been tried to solve the problem using the optimal solution method, explained before. Obtaining the next results:



Number of Delivery Warehouses	Number of Solutions	List with the Warehouse Sequence	Time for the Deliveries	Time to Generate a Solution(in seconds)
5	120	8 - 1 - 3 - 11 - 9	412,57	0,0071
6	720	17 - 8 - 1 - 3 - 11 - 9	457,98	0,0449
7	5.040	17 - 14 - 1 - 3 - 8 - 11 - 9	503,77	0,3752
8	40.320	17 - 12 - 14 - 1 - 3 - 8 - 11 - 9	540,9	3,9336
9	362.880	17 - 12 - 6 - 14 - 1 - 3 - 8 - 11 - 9	565,38	33,1693
10	3.628.800	17 - 8 - 3 - 12 - 6 - 14 - 1 - 11 - 13 - 9	618,03	650,8273
11	39.916.800	9 - 13 - 11 - 17 - 2 - 12 - 6 -14 - 1 - 3 - 8	676,87	6116,77
12	479.001.600	-	-	aproximately 6116,77*12 ≈ 20 hours
13	6.227.020.800	-	-	
14	87.178.291.200	-	-	
15	1.307.674.368.000	-	-	
16	20.922.789.888.000	-	-	

The complexity of this problem is n!, because with a set of warehouses, every possible permutation is tried, meaning that for 5 warehouses there will be 120 possible permutations. This makes the time to generate a solution also increase exponentially, until the point where this solution cannot be applied anymore, having aproximated times like 20 hours for 12 warehouses(in theory, it can end being less or more). In the following image we can see the exponential time for searching a solution from 5 to 11 warehouses:

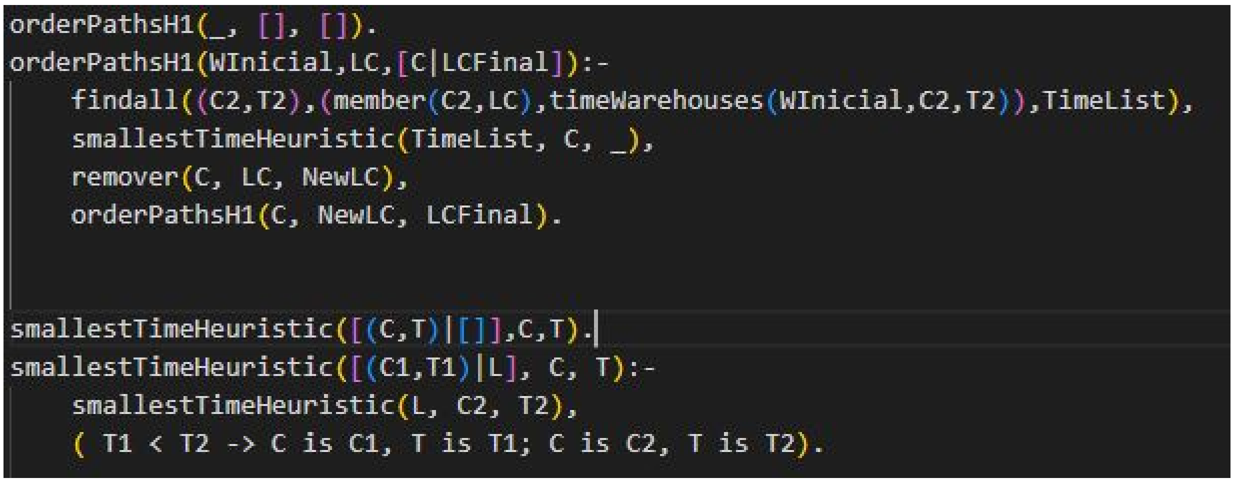


Heuristics for the quick generation of solutions

To reduce the time for finding a solutions, three diferent heuristics will be tested:

1. Choosing first the nearest warehouses.

To calculate the time, first we select the order in which we are fullfilling the heuristic purpose, and then we calculate it as if it was one the permutations solved in the previous solution. This way instead of having n! permutations we have 1. For the first heuristic it is calculated the following way:

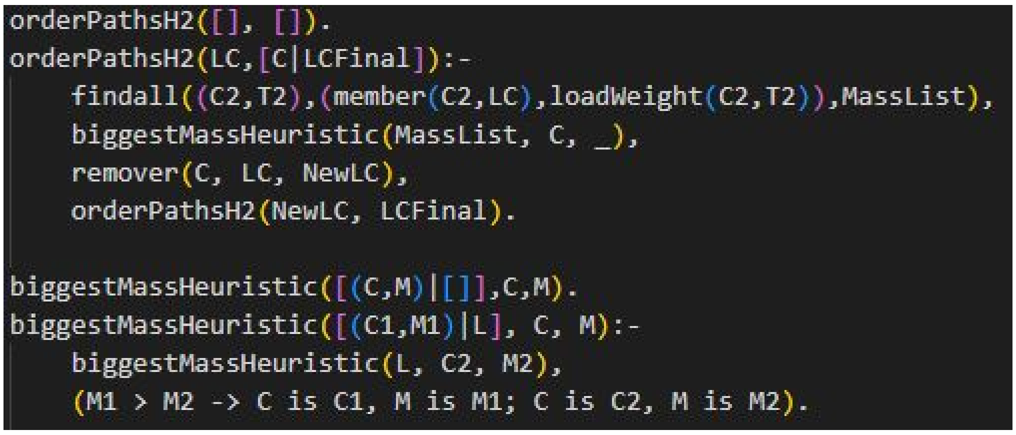

```
orderPathsH1(_, [], []).
orderPathsH1(WInicial,LC,[C|LCFinal]):-
    findall((C2,T2),(member(C2,LC),timeWarehouses(WInicial,C2,T2)),TimeList),
    smallestTimeHeuristic(TimeList, C, _),
    remover(C, LC, NewLC),
    orderPathsH1(C, NewLC, LCFinal).


smallestTimeHeuristic([(C,T)|[]],C,T).
smallestTimeHeuristic([(C1,T1)|L], C, T):-
    smallestTimeHeuristic(L, C2, T2),
    ( T1 < T2 -> C is C1, T is T1; C is C2, T is T2).
```

WIncial is the Warehouse where we start(usually 5, the main warehouse) and LC is a List of Cities, which we are going to reorganize into LCFinal with the heuristic.

We find all the times to go from the intial warehouse to the rest of them, and keep them in TimeList, then we chose the smallest one with smallestTimeHeuristic, delete it from the city list and turn it into the next initial warehouse, until we have created all the list, chosing each time the next warehouse using the heuristic.

2.Choosing first the warehouse with the heaviest delivery

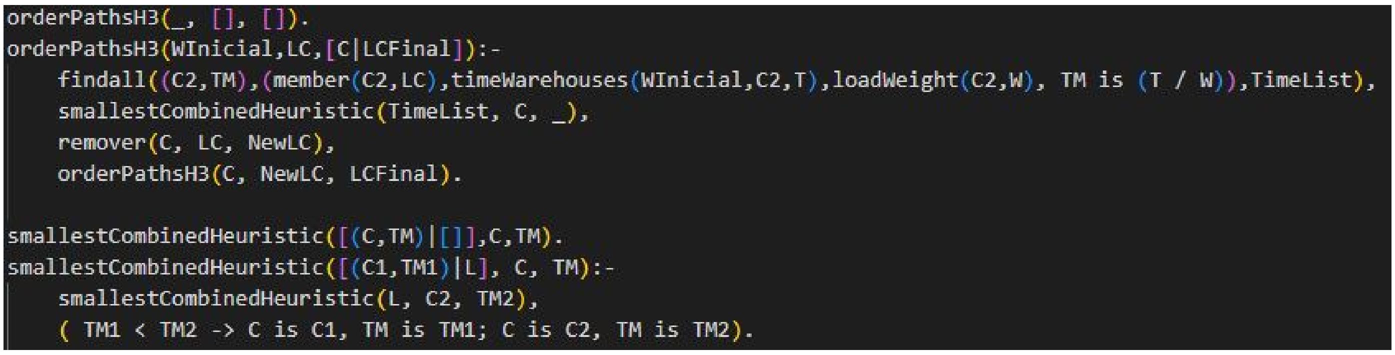
The procedure is similar to the first one:


```
orderPathsH2([], []).
orderPathsH2(LC,[C|LCFinal]):-
    findall((C2,T2),(member(C2,LC),loadWeight(C2,T2)),MassList),
    biggestMassHeuristic(MassList, C, _),
    remover(C, LC, NewLC),
    orderPathsH2(NewLC, LCFinal).

biggestMassHeuristic([(C,M)|[]],C,M).
biggestMassHeuristic([(C1,M1)|L], C, M):-
    biggestMassHeuristic(L, C2, M2),
    (M1 > M2 -> C is C1, M is M1; C is C2, M is M2).
```

Instead of finding the distance, we now have to find a list of the remaining waregouse weights, we keep them in MassList. Then we repeat the process but with a diffrent heuristic. We chose the one with the biggest mass, we delete it from the list of remaining cities and add it to the new list(the permutation from which we will get the solution time).

3.A combination of the last two heuristics

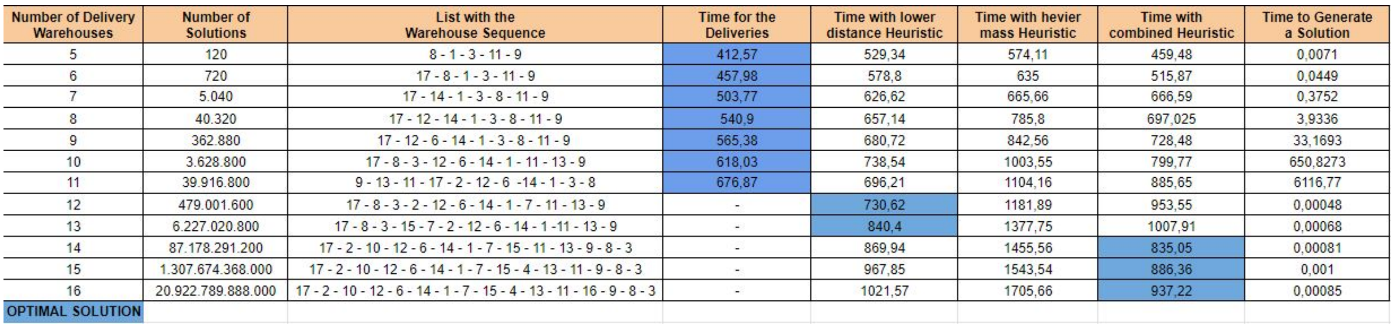
This last heuristic is a combination of the last two. This means we have to chose based on the time to the warehouse an the weight of its delivery. To fullfill that we are chosing the warehouse with best (Time / Weight).



```
orderPathsH3(_, [], []).
orderPathsH3(WInicial,LC,[C|LCFinal]):-
    findall((C2,TM),(member(C2,LC),timeWarehouses(WInicial,C2,T),loadWeight(C2,W), TM is (T / W)),TimeList),
    smallestCombinedHeuristic(TimeList, C, _),
    remover(C, LC, NewLC),
    orderPathsH3(C, NewLC, LCFinal).

smallestCombinedHeuristic([(C,TM)|[]],C,TM).
smallestCombinedHeuristic([(C1,TM1)|L], C, TM):-
    smallestCombinedHeuristic(L, C2, TM2),
    ( TM1 < TM2 -> C is C1, TM is TM1; C is C2, TM is TM2).
```

The code is a combination of the last two. We fins the Time/Weight for every remaining warehouse an keep them in TimeList, then we chose the smallest one, delte it from the city list, make it the new initial warehouse, and repeat until we have the perfect combination.

Analysis of the heuristics quality



Number of Delivery Warehouses	Number of Solutions	List with the Warehouse Sequence	Time for the Deliveries	Time with lower distance Heuristic	Time with hevier mass Heuristic	Time with combined Heuristic	Time to Generate a Solution
5	120	8 - 1 - 3 - 11 - 9	412,57	529,34	574,11	459,48	0,0071
6	720	17 - 8 - 1 - 3 - 11 - 9	457,98	578,8	635	515,87	0,0449
7	5.040	17 - 14 - 1 - 3 - 8 - 11 - 9	503,77	626,62	665,66	666,59	0,3752
8	40.320	17 - 12 - 14 - 1 - 3 - 8 - 11 - 9	540,9	657,14	785,8	697,025	3,9336
9	362.880	17 - 12 - 6 - 14 - 1 - 3 - 8 - 11 - 9	565,38	680,72	842,56	728,48	33,1693
10	3.628.800	17 - 8 - 3 - 12 - 6 - 14 - 1 - 11 - 13 - 9	618,03	738,54	1003,55	799,77	650,8273
11	39.916.800	9 - 13 - 11 - 17 - 2 - 12 - 6 -14 - 1 - 3 - 8	676,87	696,21	1104,16	885,65	6116,77
12	479.001.600	17 - 8 - 3 - 2 - 12 - 6 - 14 - 1 - 7 - 11 - 13 - 9	-	730,62	1181,89	953,55	0,00048
13	6.227.020.800	17 - 8 - 3 - 15 - 7 - 2 - 12 - 6 - 14 - 1 -11 - 13 - 9	-	840,4	1377,75	1007,91	0,00068
14	87.178.291.200	17 - 2 - 10 - 12 - 6 - 14 - 1 - 7 - 15 - 11 - 13 - 9 - 8 - 3	-	869,94	1455,56	835,05	0,00081
15	1.307.674.368.000	17 - 2 - 10 - 12 - 6 - 14 - 1 - 7 - 15 - 4 - 13 - 11 - 9 - 8 - 3	-	967,85	1543,54	886,36	0,001
16	20.922.789.888.000	17 - 2 - 10 - 12 - 6 - 14 - 1 - 7 - 15 - 4 - 13 - 11 - 16 - 9 - 8 - 3	-	1021,57	1705,66	937,22	0,00085
OPTIMAL SOLUTION							

We tried the same tests we did for the optimal solution for the 3 heuristics. It is obvious that the time for the deliveries with the heuristics is much worse than with the optimal solution, but the ecreasing of the time needed to generate a solution is inmense. For example, with 12 warehouses we would be around 20 hours to get a solution, but with the heuristics is much less than a second. This is because the complexity of the problem now is much smaller. Although it still depends on the number of warehouses, with an exponential cost is too difficult to scale, so we had to chose the optimal solutions from the heuristics tests.

Comparing the different heuristics, the first one works well above the others for tests with a small number of warehouses, although it is much higher than the optimal.

The second one is not complete enough, and his time is much worst than the optimal, and even worst when dealing with more than 10 warehouses.

On the other hand, the third heuristic works better with a bigger number of warehouses, maybe because with more warehouses and more possibilities there can be more combinations of time and weight.

Even though they are not as good as the optimal solution, it is certain that we could find an heuristic that leads to a similar time than the optimal, not dealing with the exagerated amounts of time needed to find that solution.

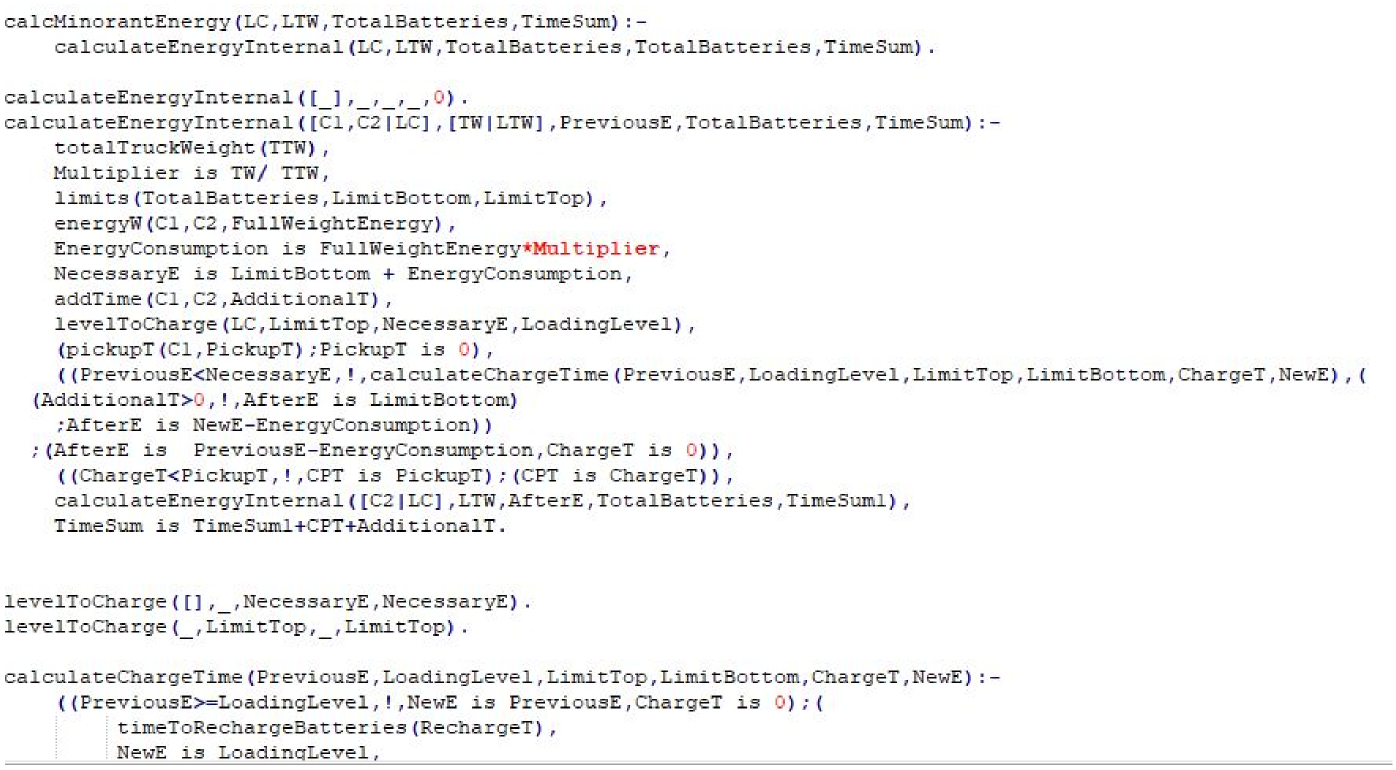
Minorantsand Majorants of the time to produce a solution

For the minorant, we have to assume that we always have the least cost possible. This is divided in two parts, first, choosing the smallest possible way time, and the choosing the smallest possible charging time.

```
calculateMinorant(LC,TotalTime):-
    sumOfWeight(LC,LW,_),
    truckWeight(TruckW),
    addTruckWeight(TruckW,LW,LTW),
    mainWarehouse(MainCW),
    append([MainCW|LC],[MainCW],TotalLC),
    totalBatteries(Batteries),
    calculateMinorantTimes(TotalLC,_,TotalWayTime),
    calcMinorantEnergy(TotalLC,LTW,Batteries,TChargingTime),
    TotalTime is TChargingTime+TotalWayTime.
```

```
calculateMinorantTimes([_],[],0).
calculateMinorantTimes([C|LC],[Time|LTimes],TotalTime):-
    findall((C2,T2,W2),(member(C2,LC),timeWarehouses(C2,C,T2), loadWeight(C2,W2)),TimeList),
    smallestTime(TimeList, CNext, T, W),
    timeWarehouses(C,CNext,FullWeightTime),
    remover(CNext, LC, NewLC),
    calculateMinorantTimes(NewLC, LTimes, TotalTime1),
    totalTruckWeight(TTW),
    truckWeight(TW),
    WToWarehouse is TW + W,
    Multiplier is WToWarehouse/ TTW,
    Time is FullWeightTime*Multiplier,
    TotalTime is TotalTime1+Time.

smallestTime([(C,T,W)|[]],C,T, W).
smallestTime([(C1,T1, W1)|L], C, T, W):-
    smallestTime(L, C2, T2, W2),
    ( T1 < T2 -> C is C1, T is T1, W is W1; C is C2, T is T2, W is W2).
```

Each time we have to chose the warehouse with the smallest time, and for the multiplier, we assume that we only have that warehouse delivery.



```
calcMinorantEnergy(LC,LTW,TotalBatteries,TimeSum):-
    calculateEnergyInternal(LC,LTW,TotalBatteries,TotalBatteries,TimeSum).

calculateEnergyInternal([_],_,_,_,0).
calculateEnergyInternal([C1,C2|LC],[TW|LTW],PreviousE,TotalBatteries,TimeSum):-
    totalTruckWeight(TTW),
    Multiplier is TW/ TTW,
    limits(TotalBatteries,LimitBottom,LimitTop),
    energyW(C1,C2,FullWeightEnergy),
    EnergyConsumption is FullWeightEnergy*Multiplier,
    NecessaryE is LimitBottom + EnergyConsumption,
    addTime(C1,C2,AdditionalT),
    levelToCharge(LC,LimitTop,NecessaryE,LoadingLevel),
    (pickupT(C1,PickupT);PickupT is 0),
    ((PreviousE<NecessaryE,!,calculateChargeTime(PreviousE,LoadingLevel,LimitTop,LimitBottom,ChargeT,NewE),(
  (AdditionalT>0,!,AfterE is LimitBottom)
    ;AfterE is NewE-EnergyConsumption))
  ;(AfterE is  PreviousE-EnergyConsumption,ChargeT is 0)),
    ((ChargeT<PickupT,!,CPT is PickupT);(CPT is ChargeT)),
    calculateEnergyInternal([C2|LC],LTW,AfterE,TotalBatteries,TimeSum1),
    TimeSum is TimeSum1+CPT+AdditionalT.


levelToCharge([],_,NecessaryE,NecessaryE).
levelToCharge(_,LimitTop,_,LimitTop).

calculateChargeTime(PreviousE,LoadingLevel,LimitTop,LimitBottom,ChargeT,NewE):-
    ((PreviousE>=LoadingLevel,!,NewE is PreviousE,ChargeT is 0);(
        timeToRechargeBatteries(RechargeT),
        NewE is LoadingLevel,
```

Conclusions

To create a working optimal solution is very difficult. The better solutions we get, the bigger is the cost.

That is why the best way possible(at least in this project) to solve the problem is to find an heuristic that creates similar solutions to the optimal. This way the time is decreased significantly, and the solution is not affected in a bad way.

We also leart that for big problems it is contraproductive to try to solve it in an optimal way. Finding a good heuristic, that takes into account the most important parts of the problem is the best way to solve a problem like this, keeping a good scalability.