FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

Artificial Intelligence - ENCS434

Project No.1 : The Single Vehicle Pickup and Delivery Problem with Time Windows (PDPTW)

Prepared by: Alaa Zuhd 1180865 Rawan Yassin 1182224
Instructor: Dr. Aziz Qaroush

Section: 1

BIRZEIT
November 28, 2020

1 ABSTRACT

The aim of this project is to implement a program that solves the Single vehicle Pickup and Delivery Problem With Time Window constraint using Constraint Satisfaction Problem(CSP) technique. This problem in general has many constraints to keep consistent at any time like capacity constraint, the vehicle have a limited capacity so it can't collect items if the total load inside it will be exceeded, also it have a constraint on the time that a request can be handled, such that any request can be collected from it's pickup location and deliver it to it's delivery location based on a time limit 'time window', and the last constraint which is axiomatic that we can't visit a delivery location before going to it's pickup location and take the request from there to deliver it.

Contents

2 INTRODUCTION

In this section, we will be explaining the single vehicle pickup and delivery problem with time window (PDPTW) in some details. This problem has become one of the most important and hard problems to solve in our life. This problem is considering as NP-Hard problem because of the amount of constraints that can have, also the need of an efficient algorithms for this problem is increased with the time , since it can make our life easier.
Actually this problem can have a various number of constraints, but here in our project, we are dealing with a few number of them like , one vehicle is available , limited capacity of the vehicle , time window constraint 'time that a location should be served in, after the early time and before the late time' ,the precedence which means first we have to pick the request from it's pickup location then we can deliver it into it's delivery location , and the last one is that every request can be handled exactly once so that a location can not be visited or served more than once.
The presence of time windows make the problem particularly complicated and considered to be NP-Hard , and an exact algorithms are too slow for large problem sizes , so in our project we will try to implement an algorithm for this problem using CSP techniques , which based mainly on the heuristics that can shrink the large space in the search. With CSP techniques we will use Back tracking search , forward checking, and arc consistency. We will be considering most constrained variable 'minimum remaining values' ,heuristic , and most constraining variable heuristic .

3 PROBLEM FORMALIZATION

In this section we will be explaining out formalization of the problem in some details .

1. **Initial State**
   All variables 'locations' are not visited .

2. **Finial 'goal' State**
   All variables are visited . 'all request have been picked up and delivered to their delivery locations' .

3. **Variables**
   Each request have two variables , one for the pickup location , and the second for it's delivery location .

4. **Domain Values**
   Each variable has a value illustrating the time of visiting this location for serving it. The domain values of a pickup location is [0-late time 'the end of it's time window'] , while the domain values of a delivery location is [the value of it's pickup location - the late time 'the end of it's time window']. This approach will help in handling the precedence constraint in the problem .

5. **Successor Function**
   Assign a value to a variable if it's not assigned 'visited' . The details of how to choose that variable will be considered in the fifth section.

6. **State**
   As we depend on backtracking in our algorithm, a stack called the 'path' in our code would be representing the state (the state means the assignment of variables)

7. **Constraints**

   (a) We can't make an assignment for a variable if the load in the vehicle will exceed the maximum capacity of it .

   (b) We can't collect a request from it's pickup location before the starting time of it's time window, if we reached into the location before that time then we will be waiting until the start of it's time window.

   (c) We can't visit a delivery location before visiting it's pickup location .

   (d) We can't visit any location after the end of it's time window.

   (e) Each location can be visited only once in the same request. (Passing through a point via the line does not mean visiting that point).

4 DATA

The input data is a file containing the maximum capacity for the vehicle in the first line , the (x,y)-coordinates of the depot in the second line. Then, each of the remaining lines contains the information of each request separated by space as follows: the (x,y)-coordinates of the pickup location , the (x,y)-coordinates of the delivery location, the load of request (first a positive load used for pickup, then the negative of the same load used for delivery), the time window (start time, late time) for the pickup , and the time window of the delivery location. We have created a class in our program that is used for random generate of data, it generates data with a probability of 0.3 that a solution can be found, and about 0.7 of the generated data are with no solution, meaning that no possible path can be found and yet keep all the constraint in-violated.

5 APPROACH

In this section, we will be illustrating the algorithms we have implemented.

5.1 HEURISTICS

Considering this problem we first need to set our heuristics that would help us in keeping the mentioned above constrains:

Minimum Remaining Values

We have depends on this heuristic in-order to select the first variable with the minimum remaining value 'time' (the one who need to finish first) , this is because if we postponed it then it's most likely to violate our constrains, and then we need to do backtracking .It returns a value equal to the variable late Time - current Time - distance from the current location to the desired location. But in case that there is not enough time to go from the current location directly to the desired location then it will return -1 to indicate that there is a violation in the time window.

Least Constraining Value

We have used this heuristic with the capacity constraint, such that it returns a value equals to the current load in the vehicle added to the load of the desired variable 'if the variable is a delivery location then it's load will be negative otherwise it's positive'. But if the value of the current load of the vehicle + variable load is more than the maximum capacity of the vehicle then it will return -1 to indicate that there is a problem with the capacity constraints.

Note : The distance was calculated to be distance = $((x_1 - x_2)^2 + (y_1 - y_2)^2)^{1/2}$ $'basedonPythagoreanTheorem'$.

5.2 BACKTRACKING SEARCH

In Backtracking we are concerned with finding a suitable place for a variable in the path 're-ordering the path'. The need for finding a new suitable place for a location is due to the fact that its time window constraint will be violated with the current ordering or any of its constraint. The backtracking pops the elements one after another from the stack until it finds a suitable place keeping all the constraint for the variable we are backtracking, then it will be added there.

For more details, if we backtrack the path until there is not element to pop from the path, then in this case we can be sure that there is no solution 'it have tried all the possible places in the path, and even if this location is the first to be visited, its constraint can not be preserved, so no solution for sure'. Also, since the number of possible orders of the path can be very large

'about size! in the worst case', so we have assigned a limited number of backtracking for each element, if this number is exceed then we can say that there is no solution for these requests. As a note, we mean by a suitable place that if we add the variable to the path with it's current order, then there must be no problem with the time window of that variable, nor the capacity of the vehicle, also there must be no violation of the precedence. In case we detect a violation in the precedence 'which will occur only when we enter a back tracking for the delivery location' then we will back track the pickup of the current delivery that we consider, then we can add the delivery into the path after it's pickup location. The reason for backtracking the pickup of that delivery is that we need to add the delivery into the path and it can't be postponed any more so we have to add it's pickup then add it indeed.

5.3 FORWARD CHECKING

After choosing a variable (location) to be visited based on the heuristics, we would check how does it affect all other unvisited locations. Meaning, if finding the next minimum location results in causing the time window constraint or the capacity constraint or the precedence constraint to be violated for any of the remaining unvisited variables, then another location would be searched for, keeping the time window constraint unviolated, the capacity of the vehicle under its limit, and the precedence unviolated, if , no other location applies, a backtracking would be implemented for the first chosen minimum location, as if we tried to postpone it, we would end in backtracking it later on for violating its time constraint.

5.4 ARC CONSISTENCY

After choosing a variable (location) to be visited based on the heuristics, we would check how does it affect all other unvisited locations. We want to detect if the current assignment will cause a problem for one more step forward than the forward checking will do. So, first we apply forward checking for the current location and as we would be decreasing the allowable time for all other unviseted location so another forward checking will be performed for all other unviseted location as a second step. If the arc Consistency failed, we would be searching for another location that preserves the arc Consistency, if no location found, we would backtrack the current minimum location.

6 TEST CASE

Following is our main interface, it allows the user to choose whether he/she wants to run the algorithm with backtracking only or with forward checking or with arc consistency. It also gives the user the chance, to choose whether he/she wants to run ready test cases (previously randomly generated), or he/she wants to try the random geneterate.

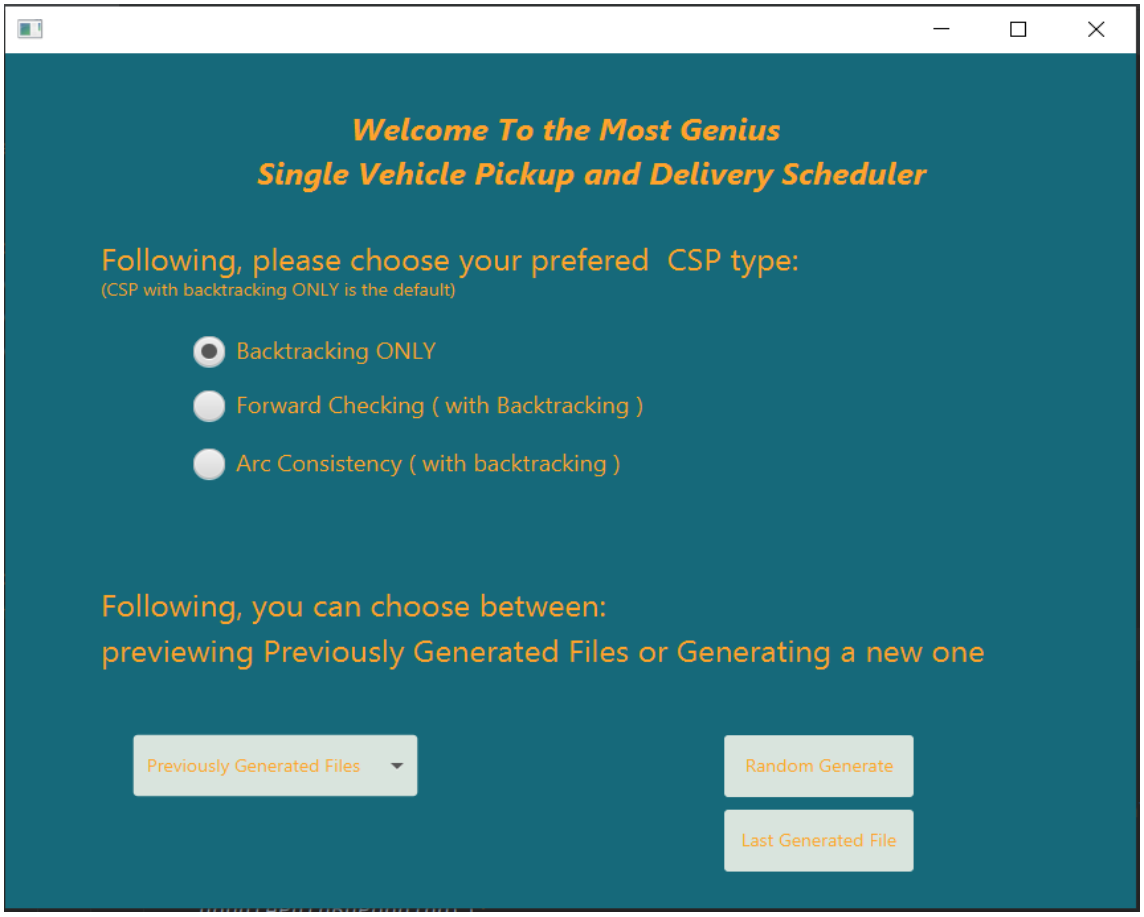

Figure 1: Main Interface

Following, we have chosen to run test case1 of the ready test cases, what appears after choosing the types the user prefers is a window containing a table showing the time when was each location visited and stating the total number of backtracking, the total needed time for that path, the total waiting time as well as the total distance as follows

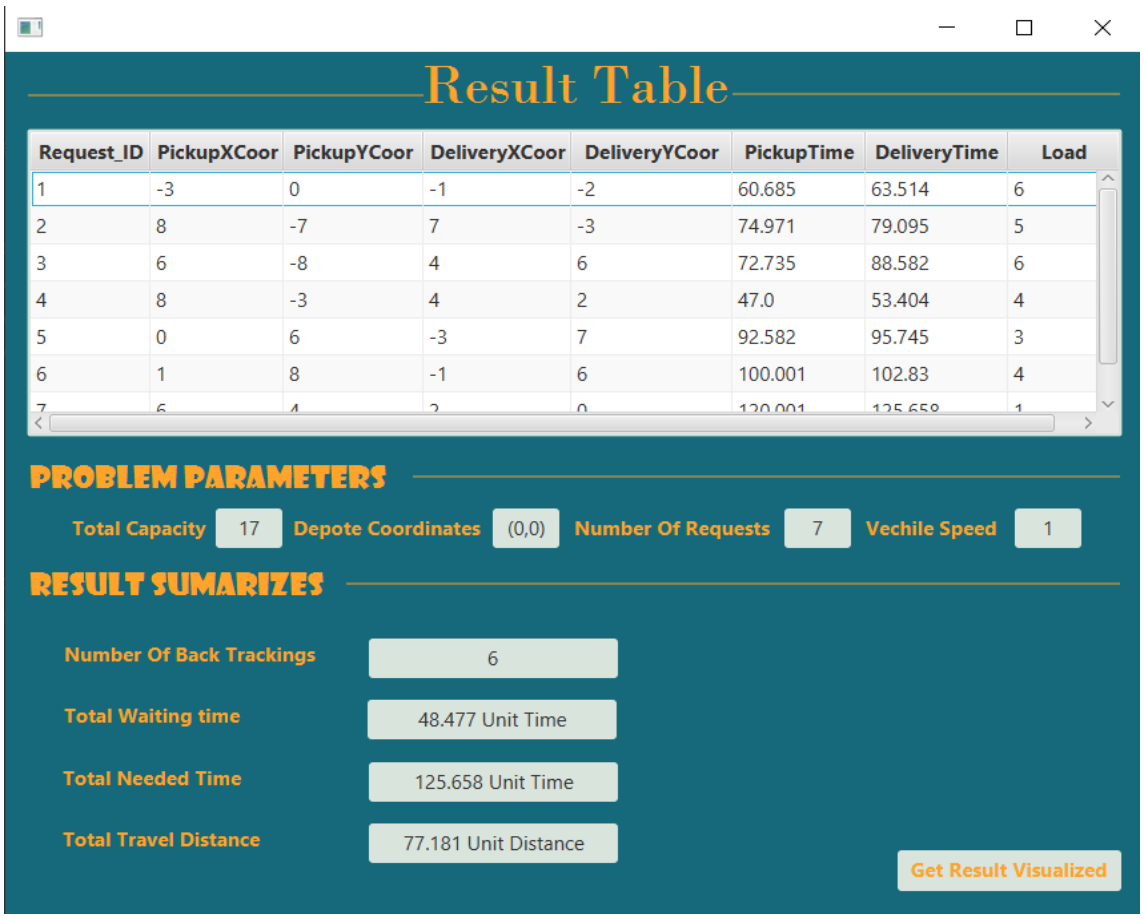

Figure 2: Test case 1 'Backtracking only'

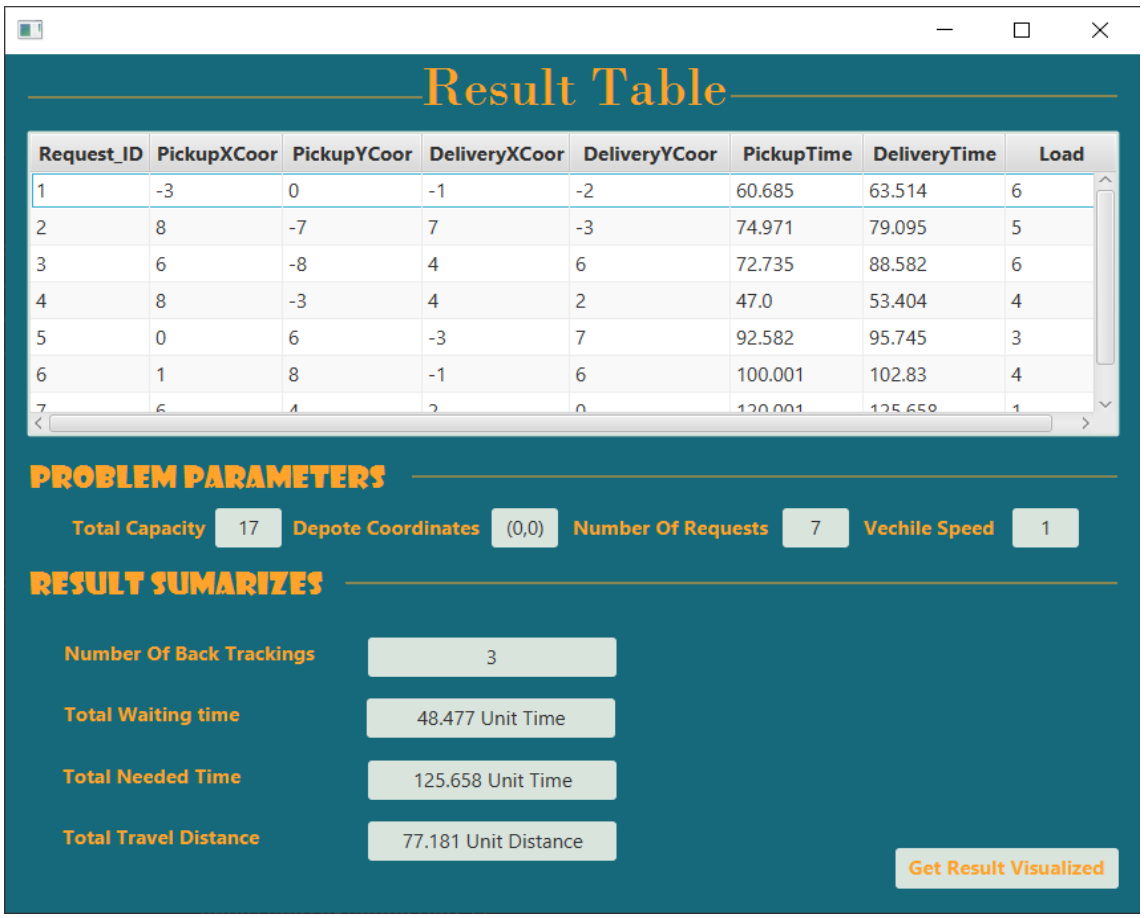

Figure 3: Test Case 2 'Forward Checking'

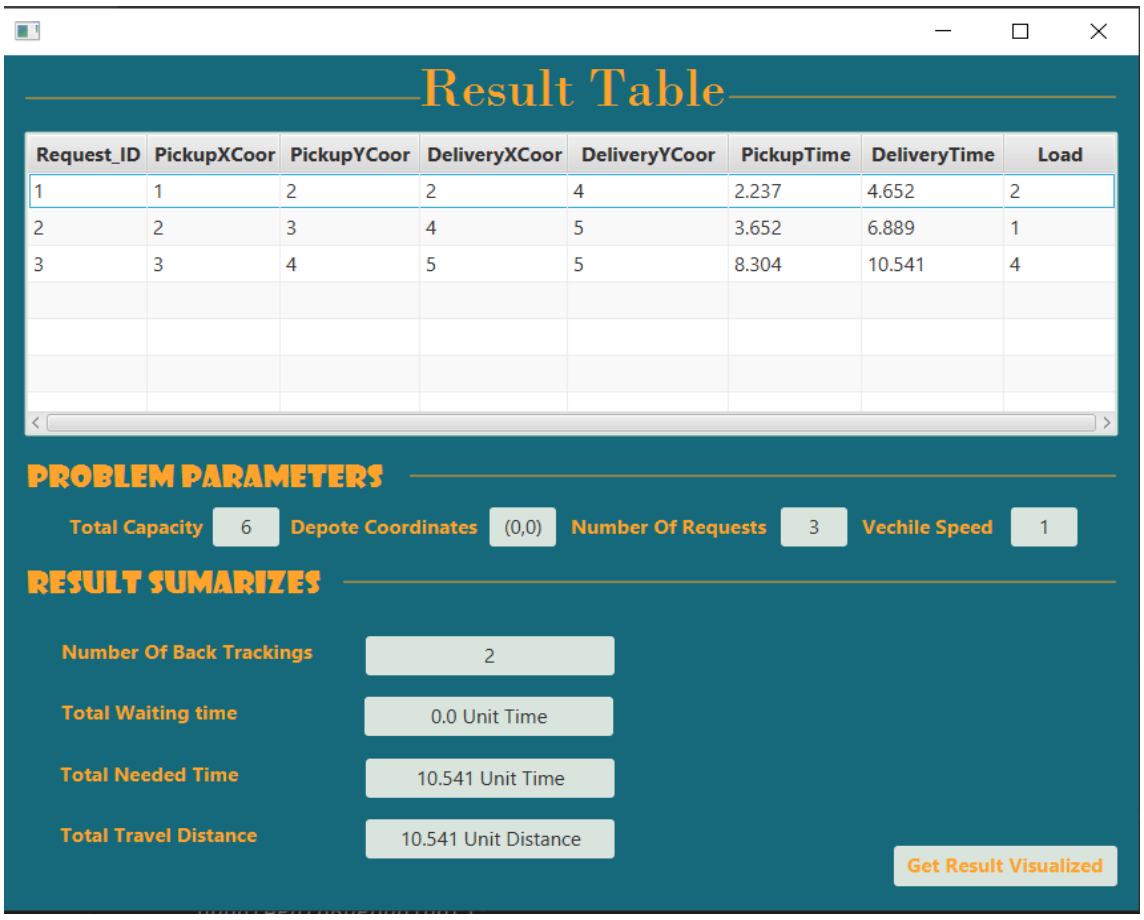

Figure 4: Test Case 3 'Arc Consistency'

Further more, if the user chooses to visualize the path, the following window will appear

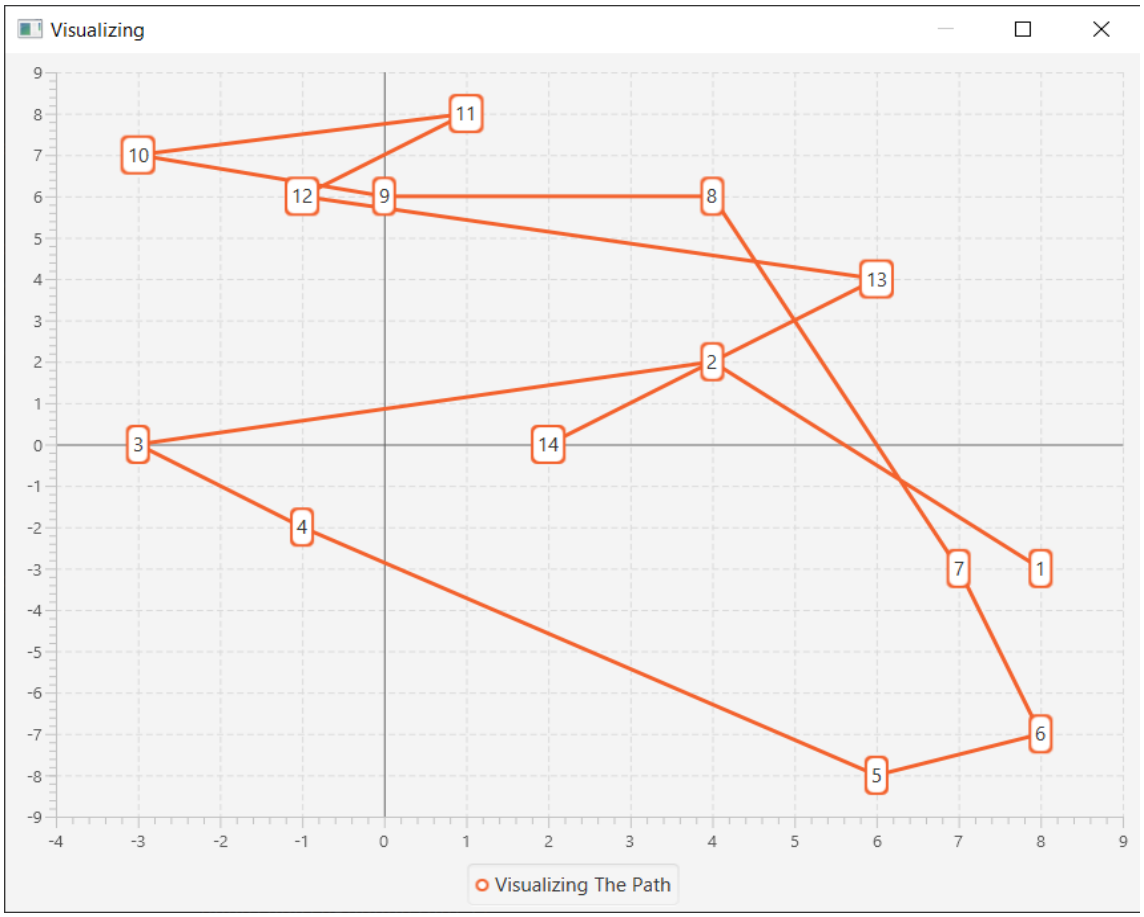

Figure 5: Visualizing

7 CONCLUSION AND FUTURE WORK

In this project we have learned a lot about CSP technique and how to think and formalize a real world problem in terms of AI field. Also, we have succeeded in implementing a program to schedule an order of collect and deliver requests with the assumed constraints(capacity constraint, time window constraint, precedence constraint), and we tried to make an interactive user interface to get a visualize of the costs and the ordering of the locations by using charts (as map) and table. We have also implemented the threading concept so that the interface is more interactive and choosing between the different available options is done easier. As a future work, this project motivated us to dive more and more in CSP technique, and other techniques in AI field, since AI is the basic unit of our life in all fields in these days, and it's the only thing that can make our life easier an easier.

8 REFERENCES

1. https://www.cnblogs.com/RDaneelOlivaw/p/8072603.html
[Accessed on 10/Nov/2020 at 1:00 pm]