Faculty of Engineering Technology
Electrical & Computer Engineering Department
ENCS3340, ARTIFICIAL INTELLIGENCE

Project Report

Prepared by:

Masa Itmaiza	**ID Number: 1200814**
Abdallah Hamed	**ID Number: 1202063**

Instructor: Dr. Yazan Abu Farha
Section: 1

Date: 10-9-2023
Time: 11:00 pm

Program Implementation:

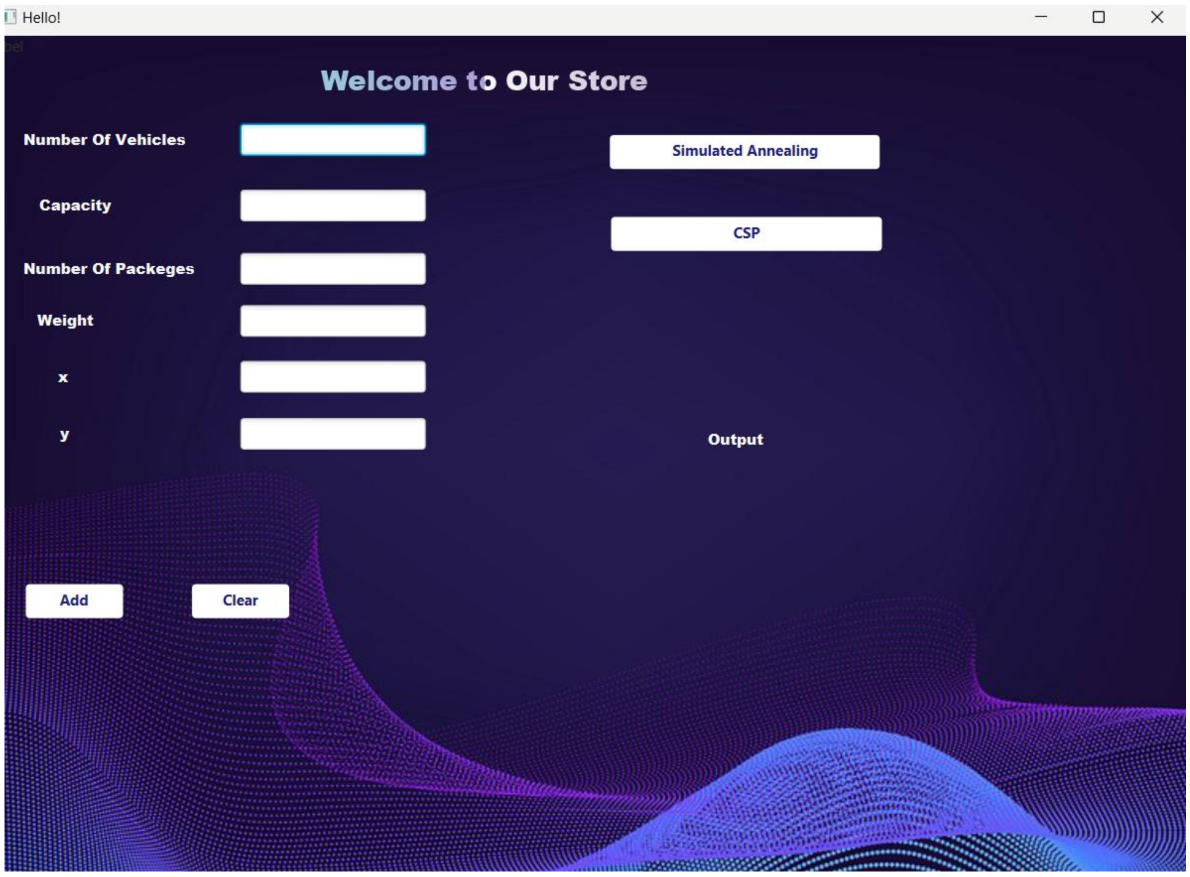
Our project is about developing a solution for a package delivery service using CSP and the simulated annealing algorithm. Every day, we receive a set of packages, each with a destination location and a weight in kilograms. Our job is to assign each package to a specific car while keeping the total weight in each car under (the capacity of each car) and specifying the route that each vehicle will take to minimize operational costs by minimizing total travelled distance.

Our goal is to load all packages into cars (if the case has a solution) and return the best solution.

Using IntelliJ 2022 Edition, we implemented our software in Java. We also use JavaFX and scene builder using css style for our UI.

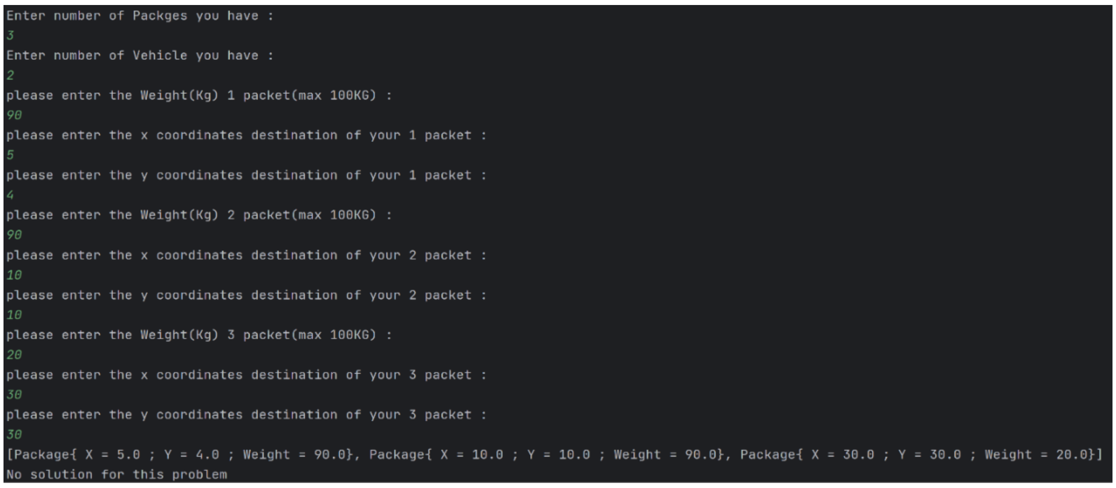
How the Program Runs

First, we have to the number of packages and cars that we have and enter all details about each package then we add packages by add button, each time user wants to add package he presses **add button** if he wants to clear the details, he entered there is **clear button**, finally the user chooses which algorithm he wants



After we enter all of the information, the application should tell me whether or not there is a solution based on the scenario we supplied. If a solution is discovered, the program will return the first solution found since, as we know, CSP is an algorithm that finds a solution, not the best one. If there is no solution for the case, it will be shown that there is no solution for this case.

For example, suppose we enter the following information. According to the limits that we set (each car carrying a maximum of 100Kg), the number of packages is three, the number of cars is two, and two of the packages weigh 90Kg and one weighs 20Kg. As the program below indicates, there is no solution to this problem in **CSP**:



```
Enter number of Packges you have :
3
Enter number of Vehicle you have :
2
please enter the Weight(Kg) 1 packet(max 100KG) :
90
please enter the x coordinates destination of your 1 packet :
5
please enter the y coordinates destination of your 1 packet :
4
please enter the Weight(Kg) 2 packet(max 100KG) :
90
please enter the x coordinates destination of your 2 packet :
10
please enter the y coordinates destination of your 2 packet :
10
please enter the Weight(Kg) 3 packet(max 100KG) :
20
please enter the x coordinates destination of your 3 packet :
30
please enter the y coordinates destination of your 3 packet :
30
[Package{ X = 5.0 ; Y = 4.0 ; Weight = 90.0}, Package{ X = 10.0 ; Y = 10.0 ; Weight = 90.0}, Package{ X = 30.0 ; Y = 30.0 ; Weight = 20.0}]
No solution for this problem
```

Simulated Annealing algorithm to find a solution by making changes to the assignment over and over again to make it better. It starts with an initial random assignment of packages, swaps packages between vehicles to create new solutions, and accepts these solutions based on acceptance probability which depends on the change in total distance and the current temperature. Over time, the algorithm cools down and it starts making it less likely to accept worse solutions. The best solution found is printed at the end when solution is near temperature.

The Project's Main Function:

- **ReadNumVehicle() :** This function read the number of cars entered by the user.
- **EuclidenDistance :** This function computes the Euclidean distance between any two places in order to compute the total distance traveled by each car.
- **TotalDistance(Vehicle v) :** This method accepts an object vehicle and uses it to determine the entire distance travelled by each car by adding the distance between any two travelled places as the car travels, and it returns the total distance from the origin point to the same position after delivering all items.
- **CheckifPossible(ArrayList<Package> arr, int numV) :** This function is used to determine whether there is enough space to add the following package in csp algorthim.
- **ReadPackages(int numP) :** This method receives each package's information from the user and stores them in an array list.
- **IsPackageLoaded(Package p) :** this function tells us the package has already loaded to car or not .

- **IsAllLoaded(ArrayList<Package> arr) :** this function tells us whether all packages have assigned to cars or not.
- **Solution(ArrayList<Package> packagee,int numV,int numP) :** This method returns a solution to the problem, which is an array list of cars, with each index indicating a car with its packages.
- **checkSpace(Package p):** this function checks if there is enough space in the vehicle for each package in simulated annealing.
- **addPackage(Package p):** Adds a package to the vehicle if there is enough space using **checkSpace** function in simulated annealing .
- **acceptSolution(double currentDistance, double newDistance, double temperature):** Determines whether to accept a new solution according to the change in distance and a temperature using acceptance probability
- **generateNeighborSolution(List<Vehicle> currentSolution):** Generates a neighbor solution by randomly swapping packages between vehicles
- **generateInitialSolution(List<Package> packages, List<Vehicle> vehicles):** Generates an initial solution by randomly assigning packages to vehicles while considering capacity for vehicles.
- **simulatedAnnealing(List<Package> packages, List<Vehicle> vehicles):** This function implements the simulated annealing algorithm by generating neighbor solutions and accepts or rejects according to certain conditions to find best solution

Data Structure Used:

To store the packages within the vehicles, we used two array lists (one for the Vehicle class and one for the package). Each vehicle, represented by an array index, has an array of packages carried by the car.

The CSP algorithm: explores all possible packages (Values) and assigns them to car (Variable). The only constraint we have is that no car can carry more than 100KG and no packages should be left unassigned. It is fine if the car(variable) does not have any packages(values) because not all cars are necessary to be used.

The simulated annealing algorithm: It begins with an initial way of assigning packages to vehicles considering the capacity and no packages should be left size and keep trying new ways for assigning packages after that decide to keep the solution, finally after finding the best solution It use acceptance probability to control how willing it is to accept worse ways.

The Heuristic of My Code:

The CSP algorithm

CSP has three heuristics that help to avoid backtracking: most constraining variable, minimum remaining variable, and least constraining value. However, because all of our variables have the same possible capacity and using any of the previous heuristics for any value or variable leads to the same result, we decide to assign variables in order.

The simulated annealing algorithm

In simulated annealing algorithm heuristic is responsible for finding a good solution, used heuristic first with finding initial solution by randomly assigning packages to vehicles, also in neighbor solution when it generates new ways for assigning packages to vehicles, in addition to temperature control by using acceptance probability equation.

Test Cases:

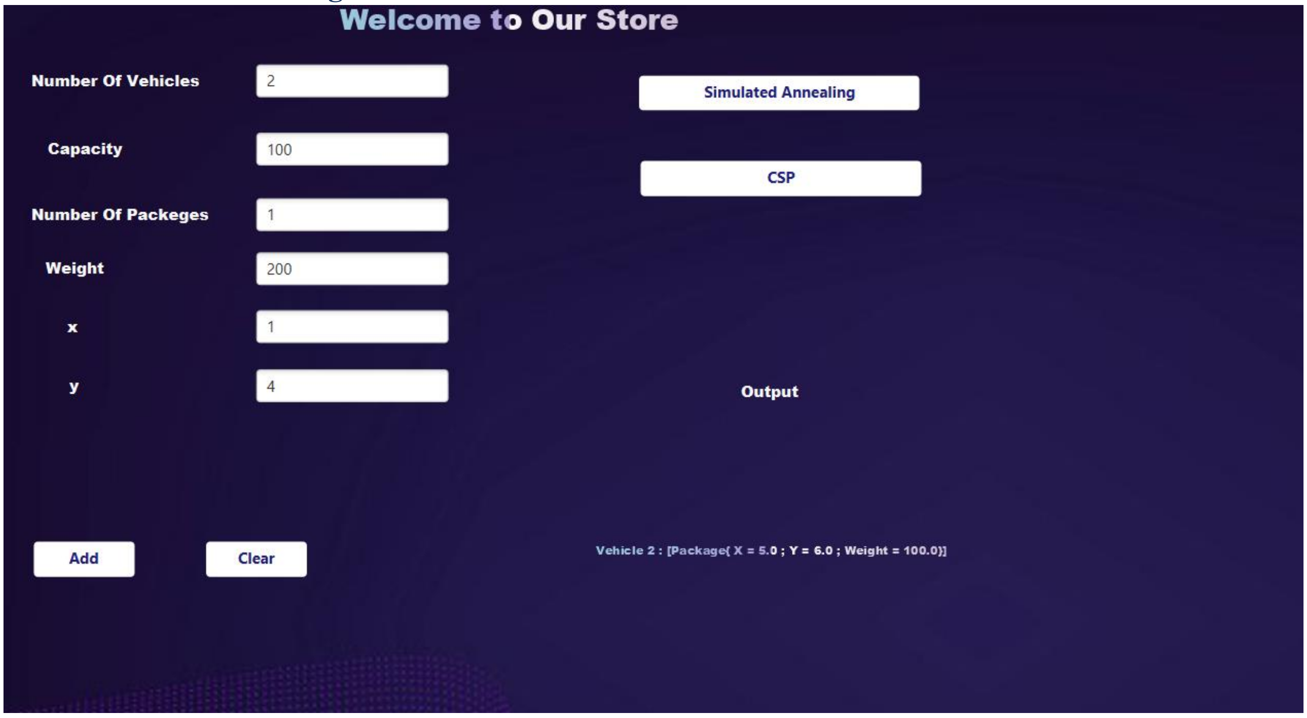
These some test cases on both simulated annealing and csp algorithm

If package weight bigger than capacity

User input 2 vehicles and capacity is 100 for each vehicles
If user entered X=5 Y=6 and weight=100 for package one
and entered X=1 Y=4 and weight=200 for package two

- **output:**

For simulated annealing:

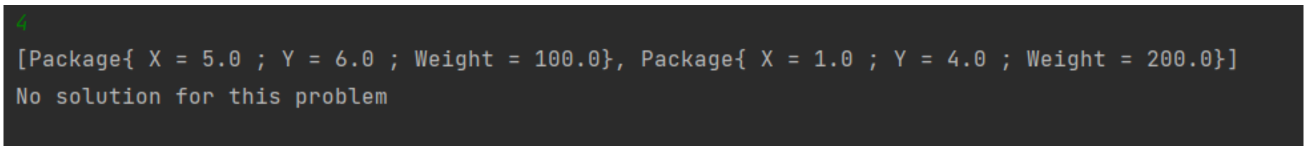


Welcome to Our Store

Number Of Vehicles	2
Capacity	100
Number Of Packeges	1
Weight	200
x	1
y	4

```
Vehicle 2 : [Package{ X = 5.0 ; Y = 6.0 ; Weight = 100.0}]
```

For CSP:

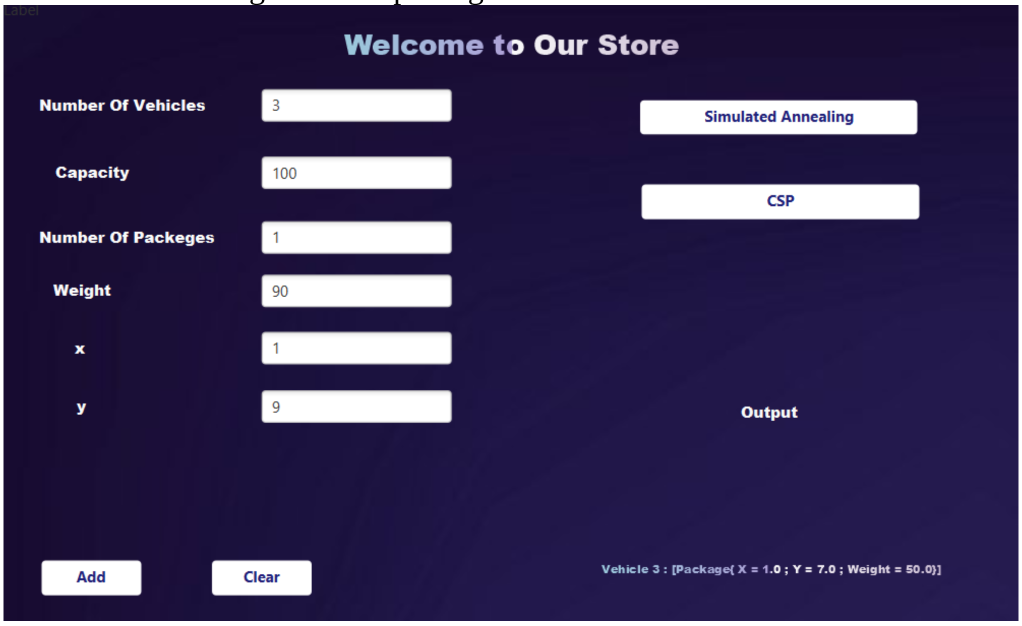


```
4
[Package{ X = 5.0 ; Y = 6.0 ; Weight = 100.0}, Package{ X = 1.0 ; Y = 4.0 ; Weight = 200.0}]
No solution for this problem
```

If package weight is fitting the capacity

User input 3 vehicles and capacity is 100 for each vehicles
If user entered X=1 Y=7 and weight=50 for package one

and entered X=5 Y=33 and weight=10 for package two
and entered X=1 Y=9 and weight=90 for package two



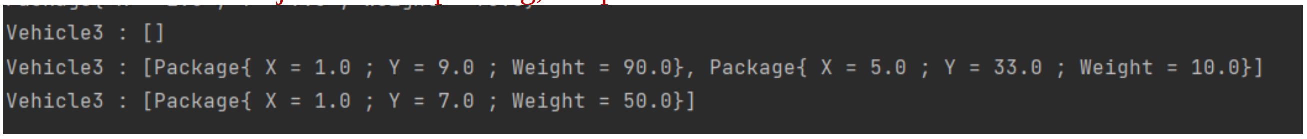
Welcome to Our Store

Number Of Vehicles	3
Capacity	100
Number Of Packeges	1
Weight	90
x	1
y	9

Vehicle 3 : [Package{ X = 1.0 ; Y = 7.0 ; Weight = 50.0}]

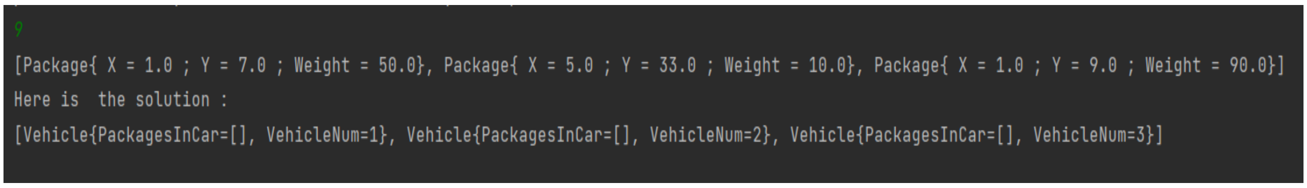
For simulated annealing:

Note: “vehicles 3” is just error in printing, 3 represents the number of vehicles



```
Vehicle3 : []
Vehicle3 : [Package{ X = 1.0 ; Y = 9.0 ; Weight = 90.0}, Package{ X = 5.0 ; Y = 33.0 ; Weight = 10.0}]
Vehicle3 : [Package{ X = 1.0 ; Y = 7.0 ; Weight = 50.0}]
```

For CSP:



```
9
[Package{ X = 1.0 ; Y = 7.0 ; Weight = 50.0}, Package{ X = 5.0 ; Y = 33.0 ; Weight = 10.0}, Package{ X = 1.0 ; Y = 9.0 ; Weight = 90.0}]
Here is  the solution :
[Vehicle{PackagesInCar=[], VehicleNum=1}, Vehicle{PackagesInCar=[], VehicleNum=2}, Vehicle{PackagesInCar=[], VehicleNum=3}]
```