Abstract

In this study, we examine a Time-Dependent Pickup and Delivery Problem with Time Windows (TDPDPTW). This problem involves maximising profit, given the constraints of multiple vehicles, time-dependent travel time with breakpoints, pickup and deliveries which have to be fulfilled in specific time windows, vehicle capacity, along with maximum trip duration constraints. We first defined the problem, inputs and solution representation. Then, we proceeded to develop the objective function and feasibility criteria. We designed an algorithm to generate basic feasible solutions and proceed to implement neighbourhood and population meta-heuristics to improve the solution. Lastly, we evaluated the performance of each of these meta-heuristics implemented to see their strengths and weaknesses.

Table of Contents

1. Problem Description

1.1 Overview

This study is motivated by last-mile delivery services which have been increasing in popularity in recent years. The problem we tackle is scheduling the pickup and deliveries of such a company in one day. The company has access to a set number of vehicles and aims to schedule a route to maximise profit. The timing of the workday is set to be 14 hours for the deliveries to be completed. The customer can specify the acceptable timing window of each pickup or delivery and they can only be served within that time slot. Each vehicle has a maximum load capacity and the weight of the items carried cannot exceed the limit. Furthermore, due to different traffic conditions at different time zones, the time taken to travel across each path will be dependent on the state of the traffic.

This real-world problem is modelled by a Time-Dependent Pickup and Delivery Problem with Time Windows (TDPDPTW). This is an NP-hard problem and an optimal solution cannot be found in polynomial-time. As such, we will be examining the performance of various algorithms and meta-heuristics to analyse their performance in solving the TDPDPTW.

2. Literature Review

Many research papers have covered Travelling Salesman Problem (TSP) or Vehicle Routing Problem (VRP) (S. N. Kumbharana et al. 2013) but there are only a few who have included the element of time-dependent travel time (B. Pan et al. 2020; P. Sun et al. 2018; M. Mahmoudi et al. 2016). The papers which have solved such problems employ a variety of neighbourhood search heuristics and population heuristics.

The purpose of this study is to evaluate the effectiveness of such algorithms implemented in the TDPDPTW problem. The specific algorithms that we will be examining are Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO).

3. Methodology

3.1. Input Processing

The input is taken in by processing the input files which are provided line by line. As per the readme file provided for the inputs, the first three lines are taken in as variables for the number of vehicles, vehicle capacity, and maximum time respectively. The next set of codes takes in the depot, pickup, and delivery nodes. For each node, the x and y coordinates are recorded. The time window for the depot node is from 0 to maximum time. For pickup and delivery nodes, the load and time windows are also recorded as per input file. As there should only be profit if the delivery is completed, the profit is set to 80 for all delivery nodes only. To form a pickup and delivery pair, a dictionary is used to simply match the pickup node to the delivery node. Finally, a dictionary is used for easy reference to all nodes, and two lists are built in the last set of input codes to store the speed and speed-choose matrix given by the input file.

3.2. Solution Representation

The solution of the question will be stored in an ordered list of lists where the list is the entire solution and each of the sublists will be the routes that each of the vehicles will be taking. The routes are shown by their node number, with 0 being the start and end of the route as that is the depot where all routes start and end. The profit of each route is included at the end of each route as well.

4. Algorithm Development

4.1. Objective Function

The original objective function is given as follows :

$$max(\sum Profits - \sum Operational\ Costs - \sum Fixed\ Costs)$$

Where profits are from the requests served, operational costs are from delivering the goods and the fixed costs arise from owning the facilities and the fleet.

In our implementation, since we are comparing between the solutions, we assume that the fixed cost is a fixed value that would be the same throughout the different solutions thus removing that from our objective function. Our objective function is as follows:

$$max \sum (Profits - End\ Time + Start\ Time)$$

Profits are from the requests served on each route and the difference between the start and end time would represent the operational cost of each route.

4.2. Feasibility Check

For the feasibility check, we apply it on each of the routes of the solution and check whether they violate the following few constraints

- Load at each node <= Capacity of vehicle
- Pickup and delivery time is within the time window given for each node
- Total time taken <= Maximum delivery time

For the following two constraints :

- All pickups are delivered
- Pickup is before delivery

They are built into the solution as we insert the pickup and delivery node as a pair to ensure all pickups are delivered and pickup would be inserted before the delivery node in the route

4.3. Basic Feasible Solution Generation

For this project, we have created two algorithms to generate a basic feasible solution. Both algorithms start off with initialising each vehicle with an empty trip, then inserting pickup and delivery pairs to the vehicles until there are no more feasible nodes. The pickup and delivery nodes are inserted based on a few criteria; feasibility, time taken to complete the delivery and profit based on the objective function.

4.3.1. Random Generation

The first algorithm that generates a basic feasible solution is a random generation. The random generation algorithm works by picking a vehicle at random and inserting the earliest feasible pickup and delivery pair. This is done until there are no more nodes left.

Pseudocode:

While there are nodes available:

    Choose a vehicle at random

    Insert earliest feasible pickup & delivery

This generation allows for a broader range of possible solutions to escape the local optimum due to the random nature after more iterations. However, this algorithm tends to generate lower objective functions with a short run time.

4.3.2. Deterministic Greedy Generation

The second method to generate a basic feasible solution is a deterministic greedy algorithm. It is based on the criteria of feasibility, the timing of delivery, time window and maximising the objective function. It selects the first vehicle and tries to insert pickup and delivery pairs based on the criteria mentioned above.

Pseudocode:

For each vehicle:

    Insert the earliest feasible pickup & delivery

    While pickup & delivery pairs can be added

Insert feasible pickup & delivery pairs based on the maximum objective function

After no more nodes can be added:

    Move on to next vehicle

This algorithm guarantees a relatively higher objective function from the start. However, the heuristics applied might not break out of the local optimum as effectively.

4.4. Local Search Operators

We have used two types of local search operators in our project, which are Intra-trip reallocation and Inter-trip reallocation. Theoretically, any pickup and delivery pair can be moved to any possible position by our local search operators so that the resulting solution is still feasible.

4.4.1. Intra-trip Reallocation and Inter-trip Reallocation

The key idea of our search operators is to remove one random pair of pickup and delivery nodes from a route, and then insert them into a different position of the same route or another random route. Due to pickup and delivery problem setting, there are some constraints for nodes removal and insertion operations: (1) The original route must have at least one pickup & delivery pair (4 nodes including two depots), so that after removal original route is still feasible (2) A pair of pickup and delivery node must be removed and inserted together, this is to ensure that the pickup and delivery activity can be achieved by one vehicle. (3) The pickup node must be inserted in front of the delivery node in a route because a vehicle must pick up the goods before delivery. (4) The inserted nodes must be between the two depot nodes. Based on whether the new inserted position is in the same or different route, we have these two graphs below representing the Intra-swap operator and Inter-swap operator.

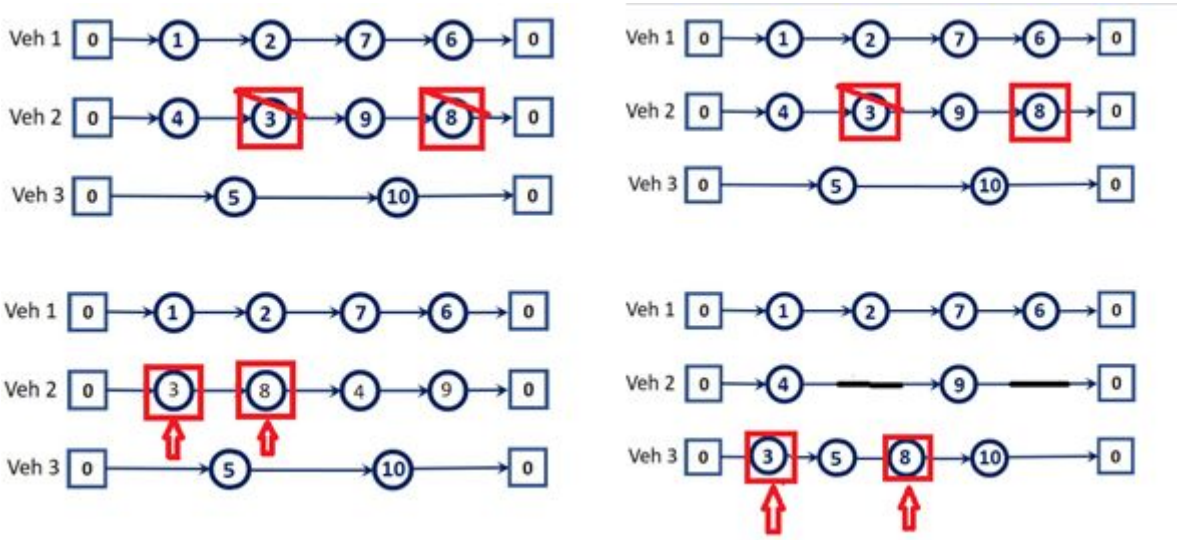

Figure 1: Intra-swap Operator Inter-swap Operator

4.4.2 Local Search Operators Pseudocode

Now we have two reallocate operators. For each operator, we have two scenarios depending on if the random node chosen is a pickup node or a delivery node. Combining all the scenarios and operators, the pseudocode of our NEIGHBOR_VRP function is shown below:

Algorithm Local Search Operators

Randomly choose a tour 1 (have nodes besides depots, nodes>=4)

1. randomly choose a tour 2

if tour 1 == tour 2:

randomly choose a position 1

randomly choose a position 2 != position 1

randomly choose a position 3

-> if it is a delivery node:

move delivery node from position 1 to position 2 (legal)

move its pair pickup node from current position to a random position 3 < position 2

-> if it is a pickup node:

move pickup node from position 1 to position 2 (legal)

move its pair delivery node from current position to a random position 3 > position 2

2. if tour 1 != tour 2:

do the same as (1) but for a different route

remembering to move the pickup and delivery pair together from tour 1 to tour 2

and pickup position < delivery position

NEIGHBOR_VRP will help us move from our initial solution to a more optimal solution. It is a key component of our SA and TS algorithm, which continuously improves our vehicle routes until the global optimal solution is found.

5. Neighbourhood Heuristics

5.1. Simulated Annealing

For this project, we implemented SA as one of the means to improve on the basic feasible solution. The initial temperature T is set to be 1000, while the ending temperature T0 is set as 10. In terms of the annealing schedule, we use a cooling coefficient of 0.9, and we perform 5 iterations before dropping to a lower temperature level. We choose the equilibrium this way as often, the improvement turns out to be zero at some temperature, therefore using an equilibrium that stops when the improvement is less significant than a threshold would often make our algorithm end immediately.

For each iteration of annealing, we simply generate a random solution and compare it with the current solution. All better solutions are accepted and worse solutions are accepted with a probability shown below, which decreases as T drops given that $\delta Profit$ is negative for any worse solutions.

$$P = e^{(\frac{\delta Profit}{T})}$$

$$\delta Profit = Profit(newS) - Profit(currentS)$$

We implemented SA with the above setup for two kinds of basic feasible solutions; deterministic greedy initial solution and random initial solution. The results are illustrated and discussed in the numerical experiment section.

5.2 Tabu Search

Tabu Search is another technique that we have implemented for improvement upon our basic feasible solution. We use a tabu list with a size of 5, and an iteration number of 2,000 and 90,000 to represent a quick search and a more comprehensive search. For each iteration, we generate new solutions from the neighbour of the basic feasible solution and store the 5 most recent solutions in the tabu list. Any new solution that is in the tabu list will not be accepted, and the list keeps rolling with the new solutions generated and removing the old ones. The best profit obtained from tabu list for our problem is at a comparable level with that of the SA approach, and one of the limitations we found with the tabu search is that when the number of iterations is small, the result could be far from the optimal solution.

6. Population Heuristics

6.1. Genetic Algorithm

Genetic algorithms have been widely used for solving scheduling and routing problems. TDPDP is a problem consisting of both vehicle routing and time-dependent scheduling. Therefore, we are thinking of using GA to solve TDPDP. However, most research uses GAs to solve traditional travelling salesman related problems. Only a few research papers use GA for solving time dependent vehicle routing problems when considering both pickup and delivery scheduling simultaneously. The most challenging part is to find a way to encode solutions and to configure genetic operators that satisfy our problem-specific constraints.

6.1.1. The Genetic Operators

Mutation: The mutation genetic operators happen when the children of two solutions are born. The percentage of children who mutated is called Mutation Rate. We have set the mutation rate to be 0.4, because a high mutation rate will help current solution to jump out of a local optimal solution, and thus a global optimal is more likely to be found. For each mutated child, the solution will be different from the crossed-over result. Some of its nodes are relocated to other positions or other routes. This is similar to our previous local search operators. Instead of one pickup delivery pair in local search, the mutation could change multiple P-D pairs. The mutated solutions should still be feasible in terms of P-D binding constraint.

Crossover: We have tried two crossover operators. The first one is a traditional crossover operator called Two-Point Crossover. Two nodes are randomly selected, the routes in

between are swapped for two parent solutions and the head and tail segments will be exchanged for two parent solutions. This traditional crossover operator does not work well with our project because the cutting of route will separate pickup and delivery-pair. The crossed-over results are not feasible anymore. Therefore we design the new crossover operators as below: (1) we randomly pick route A from solution 1 and route B from solution 2 (2) merge route A and route B into a single route (3) if the nodes exists in current route or previous routes, we remove the newly added pickup delivery pairs. (4) Repeat until all the vehicles have a crossed-over route. (5) Combine all routes for different vehicles and the new solution is a crossed-over result from its parent solutions. This crossover operator will hybrid the node sequences from parent solutions and pass them to their children. Meanwhile, all the child solutions will remain feasible after decoding.

6.1.2. Overall GA algorithm:

After setting up all the genetic operators, the overall GA framework is as below: (1) the fitness function is the same as the objective function explained in the previous section (2) The initial population is generated by the constructed algorithm described in the previous section (3) Evaluate and select top individual solutions in the population by using the fitness function. The retention rate is 0.3, which means only 30% of the initial population will be retained after selection. (4) Use crossover and mutation operators described in the previous section to generate an offspring population (5) Repeat step 3 and step 4 until the stopping criterion or certain iteration number is satisfied. The GA's flow chart of this project is shown below:

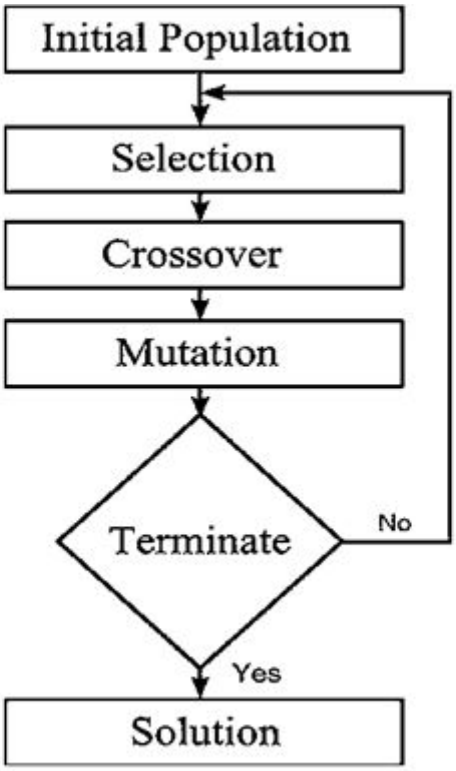

Figure 2: GA Representation

6.1.2. Observations about GA algorithm

We have observed that our GA code can find an optimal solution within lesser iterations compared with local search heuristics algorithms such as Tabu search. It means that our GA algorithm indeed passes the good traits along each population. However, results show that GA is not good at finding the global optimal solutions. If the initial population is not well designed, sometimes it will be trapped in some sub-optimal situation. I think it is due to the pick-up and delivery constraints or our conservative mutation operator. The detailed results will be discussed in section seven.

6.2. Particle Swarm Optimisation

PSO is another algorithm that has been used to solve such optimisation problems. PSO works by creating many "particles" which are represented by a velocity and location in the search space. Each of these particles is a variation of the initially generated BFS. Initially, 20 particles are generated and their velocities are calculated using this formula.

$$V\,i\,(t+1) = V\,i\,(t) + C1\ \rho1\ (P\,i\,(t) - Xi\,(t)) + C2\ \rho2\ (P\,g\,(t) - Xi\,(t))$$

In our code, velocity is defined as the node sequence differences between two solutions. Those differences or velocities will be used repeatedly to generate new solutions and populations. In our code, we have specified the two acceleration coefficients, C1 (regional acceleration coefficient) and C2 (regional acceleration coefficient), to be 8 and 12 respectively, which means that our solutions will move to goal optimal solutions faster than to regional optimal solutions.

Subsequently, the new velocities of each particle are calculated and updated based on the above formula. The current best and global best are updated and the best candidate particles are selected to repeat this process. For our algorithm, the number of iterations is set to 100 and the pseudocode of this algorithm is shown below.

PSO

```
1  Initialize a population of particles with random positions and velocities on D
   dimensions in the search space
2  while termination condition not met do
3      for each particle i do
4        Update velocity of the particle
5        Update the position of the particle
6        Evaluate the fitness f(X_i)
7        if f(X_i) < f(P_i) then
8            P_i ← X_i   // update personal best
9        end
10       if f(X_i) < f(P_g) then
11           P_g ← X_i   // update global best
12       end
13   end
14 end
```

We have observed that PSO is good at solving this TDPDP problem, because all solutions are moving to existing global optimal solutions directly. The space around existing global optimal solutions are heavily searched, but it could also be dangerous for large scale problems if the existing global solution is actually not the real global optimal solution. The detailed results from PSO algorithms will be shown in the next section.

7. Numerical Experimentation

7.1. Optimal Solution Comparison

Table 1 below shows our optimal profit obtained using the different BFS and meta-heuristics .For each of the 8 input files, the solution with the highest profit is highlighted in green.

We observe that PSO gives the best solution all the time as it has the highest profit for 8 out of the 8 inputs. Interestingly, in general, the meta-heuristics algorithm coupled with greedy BFS will result in the highest profit. We also observe GA giving the next best solution most of the time.

The Tabu search algorithm is repeated using a higher number of iterations. After some testing, we found that increasing the number from 2,000 to 90,000 gives better profits most of the time (6 out of 8 inputs using Random BFS and 7 out of 8 using Greedy BFS).

Optimal solutions using **Random** Basic Feasible Solution Generation for initial solution					
Input	Simulated Annealing	Tabu Search (2000 iteration)	Tabu Search (90000 iteration)	Genetic Algorithm	Particle Swarm Optimization
10A	318.995518	273.3837167	309.4580966	318.4581235	407.7786514
15A	565.9595963	534.505016	544.1495478	493.3872454	646.9705445
20A	647.4857691	573.6517198	605.860086	213.6983238	621.5678754
25A	804.6550418	679.8592716	614.0801863	658.0403249	837.8801668
30A	902.9423463	702.2885724	716.9218997	791.6079994	869.2388756
35A	1207.299031	963.3498061	880.2324821	1132.615268	1152.036296
40A	1315.005152	1070.295572	1178.207969	1157.326271	1162.805403
45A	1473.443685	1191.64217	1227.582412	1177.169196	1342.02701

Optimal solutions using **Greedy** Basic Feasible Solution Generation for initial solution					
Input	Simulated Annealing	Tabu Search (2000 iteration)	Tabu Search (90000 iteration)	Genetic Algorithm	Particle Swarm Optimization
10A	309.4580966	276.217611	309.4580966	331.9814993	417.982333
15A	569.9827819	515.6942423	553.1305157	570.672212	648.8763642
20A	641.1488201	551.9027483	545.6605573	420.2106592	667.6770377
25A	813.8151311	773.7052232	795.1705939	813.8151311	866.5675661
30A	935.8600357	872.1926439	918.5909972	922.1282826	1004.850023
35A	1172.769639	1137.555469	1160.162256	1172.769639	1261.320558
40A	1324.19416	1254.330797	1339.709244	1316.351044	1388.378864
45A	1498.377637	1442.653256	1479.225979	1498.377637	1577.201851

Table 1 : Optimal Solutions

7.2. Local Heuristic Evaluation

Below are some sample outputs that we obtained for the input 15A/30A. We observe that with a deterministic greedy initial solution, SA only manages to achieve a small improvement as the initial profit is already at a high level. However, with a random initial solution, SA performs better and bigger achievements can be obtained, and interestingly, making the final profit for both types of initial solutions after SA being implemented to be approximately the same. Therefore, SA is not dependent upon the initial solution and only relies on the number of iterations, i.e. temperature and equilibrium setup. This observation echoes with the notion that SA could enable us to escape from local neighbourhoods and find an optimum in the end. As a comparison between SA and TS, we can see that SA outperforms TS in both the final profit and the number of feasible solutions generated. We notice that the profit obtained by Tabu search is mostly zero, especially for a random initial solution, and this could be because the algorithm is trapped in a neighbourhood where not many feasible solutions can be generated, especially since the PDPTW has many more constraints, and is unable to escape from this neighbourhood. This has also been reflected to be one of the shortcomings of TS by others.

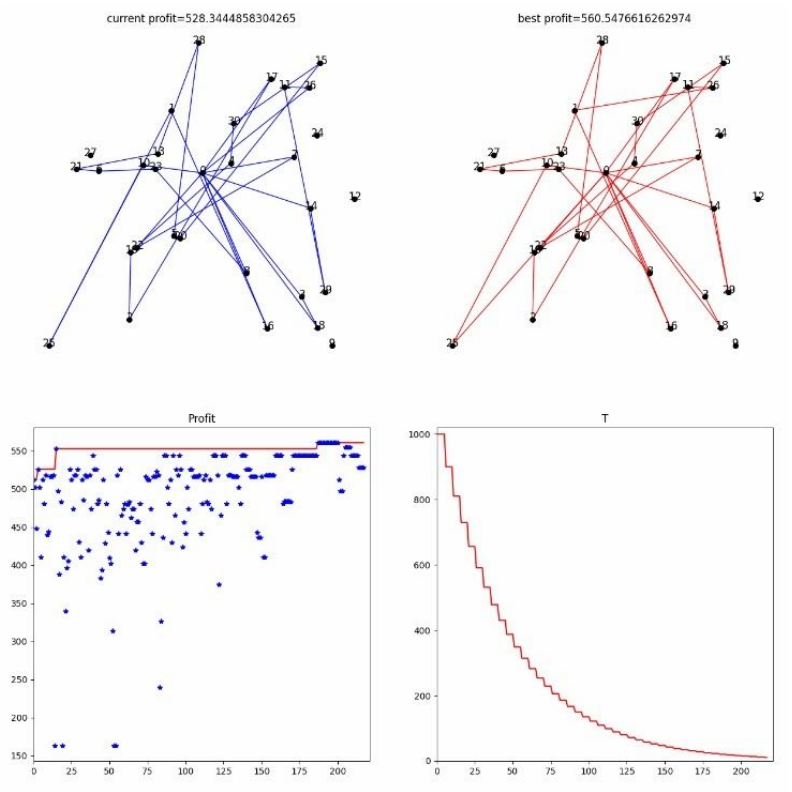

Figure 3 : Simulated Annealing of Problem 15A with Random BFS

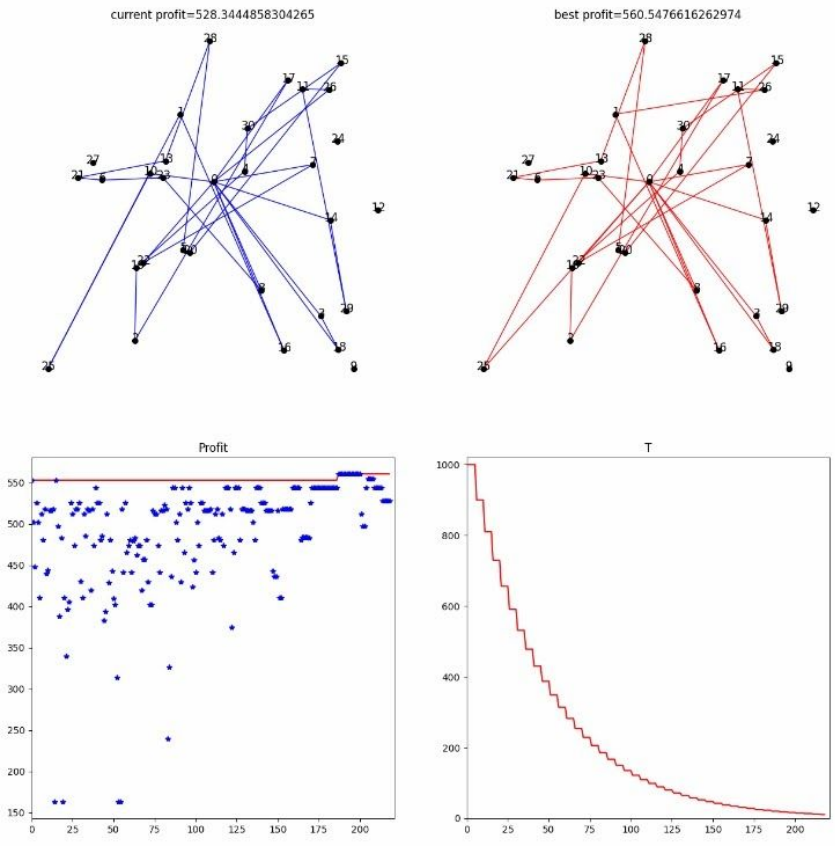

Figure 4 : Simulated Annealing of Problem 15A with Greedy BFS

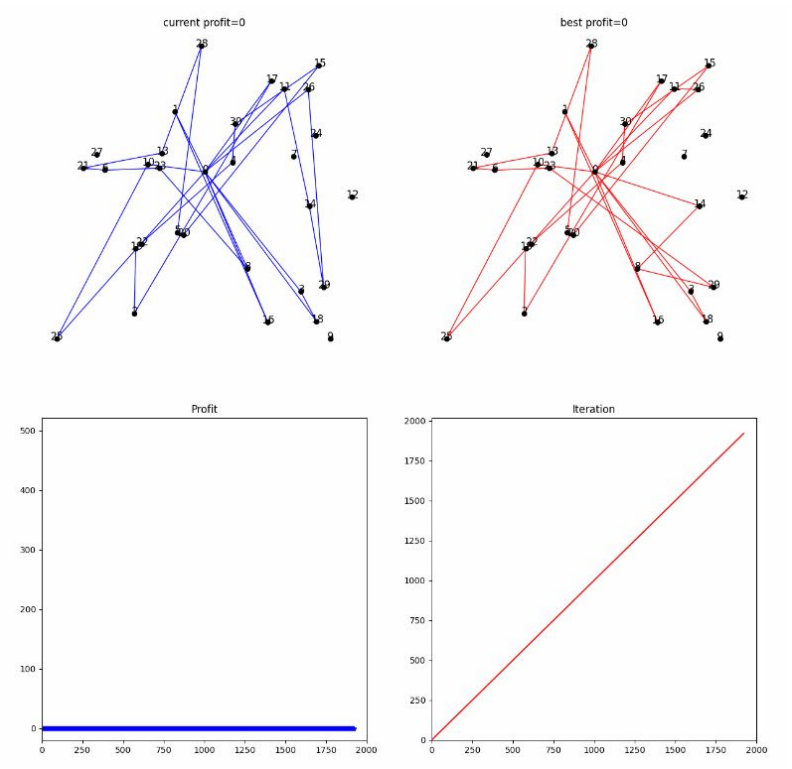

Figure 5 : Tabu Search of Problem 15A with Random BFS

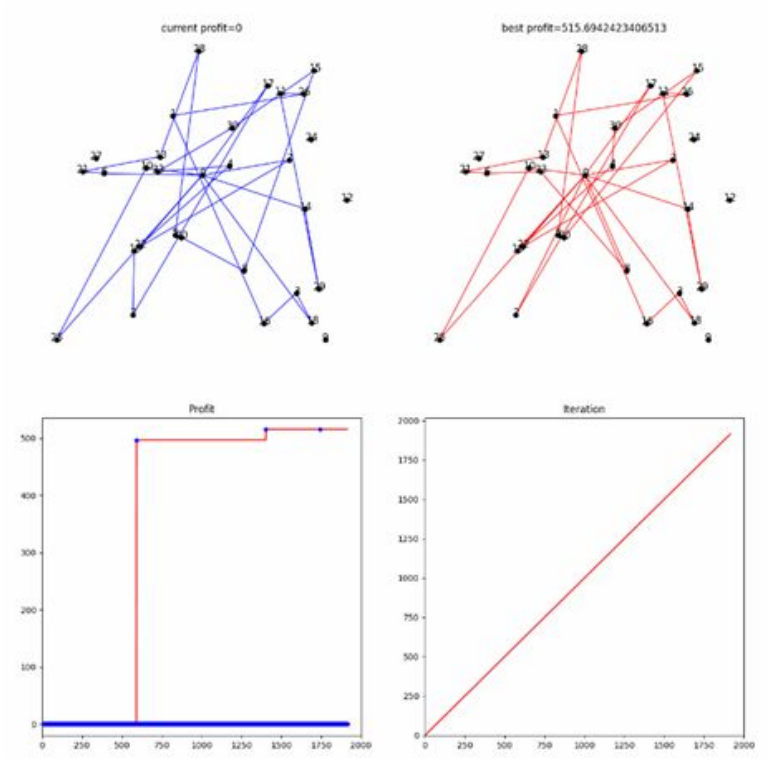

Figure 6 : Tabu Search of Problem 15A with Greedy BFS

7.3. Population Heuristic Evaluation

Examining the results from our numerical experimentation, the two population heuristics applied seem to do better than neighbourhood heuristics in general. We theorise that due to starting with more variations of the basic feasible solutions at the start, both GA and PSO are able to break free of the local optimum very easily. Subsequently, the algorithm will focus generating new solutions for the global optimal solution but still continue to search through other local optimal solutions. This gives other neighbourhoods a larger chance to surpass the previous global optimum and generate a better solution overall.

Comparing GA and PSO, PSO clearly performs better than GA. This is probably due to how the mutations for GA and velocity for PSO is generated. Due to the strict constraints of the PDPTW, PSO is able to prioritise which solution to iterate on more efficiently than GA.

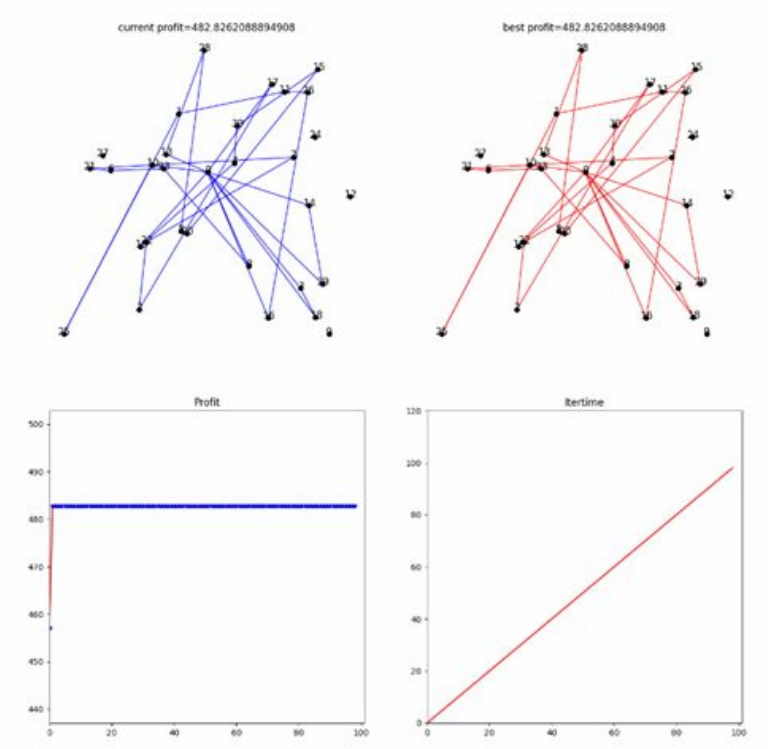

Figure 7 : Genetic Algorithm of Problem 15A with Random BFS

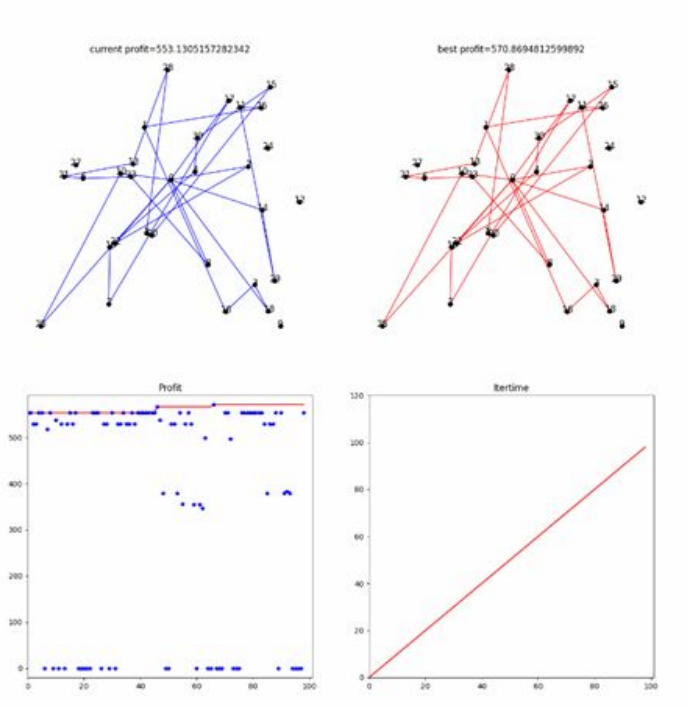

Figure 8 : Genetic Algorithm of Problem 15A with Greedy BFS

8. Conclusion

In conclusion, we have developed two methods of generating a basic feasible solution, either by Random or Greedy method. We observe that the greedy BFS will usually give a better initial solution than random BFS, but does not guarantee that it can give the best solution after improving with a meta-heuristics algorithm. We also implemented different local and population search meta-heuristics to improve the BFS and found that greedy BFS coupled with PSO will give the best solution most reliably.

Hence, a company specialising in last-mile delivery services can model the delivery problem as a Time-Dependent Pickup and Delivery Problem with Time Windows (TDPDPTW), to optimise their delivery. Based on the findings of this report, the company is recommended to focus their research on optimising this problem using population search heuristics such as

Particle Swarm Optimisation or Genetic Algorithm, with a deterministic greedy algorithm to find the initial solution.