ANT COLONY APPROACH

FOR

MULTIPLE PICKUP AND MULTIPLE DROPOFF

By

GORTHI VENKATA SREERAM PHANI SAI

Bachelor of Technology in Computer Science

GITAM University

Visakhapatnam, Andhra Pradesh, India

2009 - 2013

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 2017

ANT COLONY APPROACH

FOR

MULTIPLE PICKUP AND MULTIPLE DROPOFF

Thesis Approved:

Dr. Johnson P Thomas

Thesis Adviser

Dr. K M George

Dr. David Cline

ACKNOWLEDGEMENTS

I would like to take a moment to express my sincere gratitude to my advisor Dr. Johnson Thomas for his support, patience and continuous guidance throughout my work. His expert suggestions and valuable feedback had helped me to achieve my goal.

I would also like to thank Dr. David Cline and Dr. K. M. George for being very supportive and being part of my committee.

My sincere thanks to the Ruutdrop company for providing valuable feedback and continuous assessment on my project.

Last but not least, I would like to thank my parents for everything. They have made me into the person I am now. Their hard work and motivation have paved the path for me to achieve my goals.

Name: GORTHI VENKATA SREERAM PHANI SAI

Date of Degree: DECEMBER 2017

Title of Study: ANT COLONY APPROACH FOR MULTIPLE PICKUP AND MULTIPLE DROPOFF

Major Field: COMPUTER SCIENCE

Abstract:

The Multiple Travelling Salesman Problem, popularly known as MTSP is an NP-hard problem. MTSP is a well-known combinatorial optimization problem in which more than one salesmen visit all cities only once and return to the depot. In our problem, we apply the MTSP algorithm to multiple drivers picking and dropping packets at multiple locations and the drivers not returning to the starting location. There are no exact solutions for solving this combinatorial problem that can guarantee to find the optimal route within a reasonable time. A meta-heuristic algorithm, Ant Colony Optimization (ACO) is used as a base for our solution construction for different variations of the problem such as handling multiple pickups and multiple drop-offs using a single driver, multiple drivers, drivers starting at different times, and drivers available for different times. The goal is to maximize the number of goods delivered while minimizing distance (or time) within some threshold limits. The results are compared to existing algorithms like Brute-force approach and Nearest Neighbor algorithms. Our results show that the proposed ant colony algorithm achieves better results or at worst identical results to the Brute-force approach.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

The Multiple Travelling Salesman Problem (MTSP) is a generalization of the Travelling Salesman Problem. MTSP has many applications in the real world, such as crew scheduling, school bus routing, interview scheduling, and the design of global navigation satellite surveying networks [1]. These kinds of problems are frequently encountered in logistics. Finding efficient routes for different salesmen (vehicles) to serve multiple locations has been studied over several decades in logistics. If a company can reduce the route length traveled by individual salesmen, or reduce the number of vehicles needed to serve all locations, it will be able to service a large number of customer requests with minimal cost. The Multiple Travelling Salesman problem involves multiple salesmen visiting cities which are geographically dispersed only once and returning to the initial starting point. Within this field, many variations have been researched using different constraints such as time windows, vehicle capacity, delivering and picking up goods, and open systems where drivers need not return to the initial pickup location. Due to its economic importance and a wide range of applications, MTSP research has grown for many decades. Problem variations typically involve finding the minimum cost of a total

tour, finding the minimum number of vehicles for covering all the locations, etc. The cost can be defined in many ways, such as the distance between cities, time, and capacity. In the cases mentioned above, only one objective function exists, and optimization is performed based on that objective function.

In MTSP, there exists more than one vehicle to serve the given location in delivering or picking up goods. In the variation that this thesis considers, the vehicles need not return to the starting location (i.e., initial pickup location after it serves all the pre-determined locations assigned to it). Once the task is completed by the vehicle, it can go to any location where it can find new requests to take. In this variation, the cost is related to two parameters, namely, vehicle occupancy and distance between locations. The optimization should be performed based on the two parameters listed above making this combinatorial problem NP-hard as well as a bi-criterion problem. The selection of the next location from the current location is defined based on these parameters. Based on these requirements, an optimal route is built using meta-heuristic algorithms.

1.1 Background:

The Multi Travelling Salesman Problem is an extension of the Travelling Salesman Problem which is one of the best known NP-hard problems. There are many real-world applications in which MTSP plays a major role [1]. For example, MTSP is used in genetic engineering to minimize the length of DNA, in spacecraft to minimize fuel combustion, and in the design of global satellite systems. MTSP also plays a large role in road networks in designing routes for school buses, emergency services, traffic controls and logistics.

1.2 Motivation

Many MTSP variants have addressed the problems of handling different constraints that are mentioned above. Our current problem relates to the capacities of vehicles. In this work, there are multiple vehicles with varying capacities that start at different locations instead of starting at a single depot. Vehicles do not drop any goods at the starting point; they only pick up goods from that location. Once the vehicle is loaded with the goods, MTSP handles the construction of the route. At each point along the route, a vehicle may pick up or drop off boxes or do both. Vehicles do not pick up any goods at the end point of the graph, and they do not return to the initial starting point.

The goal of this work is to maximize the number of delivered boxes while minimizing the distance (or time).

1.3 Outline of existing work

MTSP can be defined as follows: given *n* cities and *m* salesmen starting at a given location (i.e., depot), all the cities must be visited at least once by *m* salesmen with minimal total distance. Each salesman should visit a city, which has not visited by the other salesman. MTSP determines the route for the salesman with the minimal distance, to visit all cities. The factors that need to be optimized can be the distance to be traveled, time, or capacity. [1]. There are different MTSP variants such as single and multiple depots, number of salesmen, and time frame. The heuristic and metaheuristic algorithms that are used for handling such variants are Greedy Algorithm, Genetic Algorithm [3] [10], Ant Colony Optimization [11], and Particle Swarm Optimization [10]. These algorithms generate a feasible route based on the distance that each salesman needs to travel. Also, these

algorithms usually consider single objective functions, namely, distance, time or minimizing the number of vehicles.

The goal of our work is to maximize the occupancy and minimize the distance to travel. Existing works have looked at objective functions such as minimizing the total distance travelled by individual salesmen [3], vehicles ending at a special node instead of returning to the depot [5], number of vehicles that are required to complete a task [4], assigning vehicles based on road capacity [2] etc. Existing work does not address our problem of maximizing the deliverable goods while minimizing the distance traveled. Hence, a new solution is required to address our problem.

1.4 Outline of proposed work

In this problem, we are handling two parameters, namely distance (or time) and the capacity of the vehicle, which makes the problem a bi-criterion problem. Some solutions have been proposed for MTSP as outlined above. We chose the Ant Colony approach because Ant Colony Optimization has an inherent parallelism and can rapidly discover good solutions based on positive feedback. Ant Colony Optimization is also adaptive and works efficiently for dynamic requests in polynomial time.

We modified the ant colony optimization algorithm to fit our problem. The regular way that an ant selects its next node is manipulated, and it selects based on the objective function defined above.

1.5 Outline of the Thesis

The rest of the thesis document is outlined as follows: Chapter 2 describes various research works that are related to the work of the thesis. Chapter 3 outlines the deficiency of existing work and provides the detailed solution to our approach. The proposed algorithms are simulated and the results are presented. Chapter 4 concludes the thesis with suggestions for possible future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Review

Different variations of the Multiple Travelling Salesman Problem are surveyed in [1]. The different variations listed in [1] are single vs. multiple depots [5] along with fixed and non-fixed destinations, number of salesmen, fixed charges, and time windows. Each variant is further modified according to the needs of real-world problems in designing applications. MTSP is applied in various routing and scheduling applications such as print press scheduling, crew scheduling, school bus routing, and hot roll scheduling. [1]. Another variant of MTSP is the time frame, i.e., MTSPTW – Multiple Travelling Salesman with Time Windows. Based on this, research has been done for finding the minimum number of vehicles needed to perform pickup and delivery requests in a given time window using precedence graphs [5]. This research is done primarily on vehicles that do not return to the depot after the deliveries are completed and end up at a special node. Asken, Ozyurt, and Aras in [5] developed a new Open Tabu Search algorithm for handling this problem. Fixed destinations are those where the salesman returns to the same depot after visiting

all the cities.

MTSP also has a wide range of important applications in the logistics field. Based on the type of goods they carry and on handling the pickups and deliveries there are different variants of MTSP. Some works have looked into multiple pickups, and deliveries using simulated annealing and ejection pool algorithms along with node exchange and, node relocation heuristics [6]. Handling the deliveries and pickups at the same location involves issues such as the load shuffling problem [6]. Due to this issue, there are again several variants in the problem which carries the deliveries with last-in-first-out loading [7] and first-in-first-out loading [8]. There are other variants where only the deliveries are carried out first and then requests are taken for picking up goods [6]. Other work addressed splitting the tasks where the customer is visited twice for handling requests [9] using a local search with a relocate operator, relocate split operator and a hybrid heuristic algorithm.

All these variations of MTSP, TSP, and VRP optimize a single objective function either by time, distance or type of delivery. However, there are very few works that address multiple objective functions. One such work addresses the issue of optimizing both driving time and energy consumption which are inversely proportional to each other [10] using a pseudo-polynomial time algorithm with vertex labeling algorithm.

In MTSP, if the deliveries and pickups are carried at the same time, with the same vehicle, some issues need to be considered. One such issue is the Load Shuffling Problem [6]. This problem can be defined as follows: when the vehicle handles both the pickups and deliveries in any order, there may arise a situation where the delivery goods are

inaccessible in the vehicle. This shuffling involves time spent in ordering the goods at every stop.

2.2 Existing Algorithms on MTSP

The algorithms mentioned below are some of the algorithms which are used for different MTSP problems.

2.2.1 Greedy Algorithms

This classic algorithm approximates the shortest distance that covers all the cities for a single salesman. First, all the edges are taken into the solution space and sorted. Once they are sorted, the algorithm starts constructing the route based on the shortest distance repeatedly until it covers all the nodes in the graph [11]. The algorithm is checked for both symmetric and asymmetric TSP problems based on the domination number and proved that the results are unsatisfactory because it generated the worst tour [11].

2.2.2 Nearest Neighbor Algorithm

The Nearest Neighbor (NN) algorithm starts the tour from a given starting point i and finds the nearest neighbor j from i ($i \neq j$). The tour continues until all nodes in the solution set S are visited exactly once [11]. Repeated NN (RNN) algorithm works similar to the NN, but RNN constructs the route by taking every node in the solution set as a starting point and finds the routes. The best route to the nearest distance is selected among the generated routes [11]. Both NN and RNN are analyzed with the domination number approach for symmetric and asymmetric TSP problems, and the results are obtained are not desirable since it generated the worst tour when $n \geq 2$ [11].

2.2.3 Genetic Algorithm

The algorithm initially generates the population of "chromosomes" which represents tours and evaluates the fitness for each of them. By selecting two chromosomes randomly from the parent population, it generates two offspring using process called selection, crossover, and mutation which are inspired by biological processes. A fitness function is maintained to guide the search process in the solution space of chromosomes. The old population is replaced by the new population, and the fitness is evaluated again. The search process continues till the best set population is created [14]. The list of tours is taken as the population, and the parents are selected from the population to create new child tours. The search continues until near optimal solution is obtained. The creation of new child tours and comparing them with existing tours becomes complex with increase in population size.

2.2.4 Gravitational Emulation Local Search Algorithm (GELS)

This GELS algorithm [12] is based on a local search using gravity and velocity. Gravity helps in attracting objects to each other. A heavier object has more gravity and attracts lighter objects. Each objective function is represented by a mass, and the solution with the highest mass is the best solution. In MTSP, all the cities are divided into a different group, and each group is considered as a TSP problem. Each group has different neighbors, and each neighbor is determined by the distance and the direction of the neighbor solution. The next city is selected based on the nearest distance, and with the highest velocity.

2.2.5 Ant Colony Optimization

This algorithm [13] uses the behavior of natural ants for finding optimal solutions. Ants lay pheromone trails along their route while searching for food. These pheromone trails tend to evaporate slowly. The shorter distances tend to have more pheromone deposited along their routes and are therefore likely to be chosen by other ants. This algorithm gives the solution when an ant finds a good route to the food using positive feedback. The solution for the current problem is based on ant colony optimization. In MTSP, each ant (drivers) traverses through the cities and selects the next neighboring city based on heuristics. The ant will either select the nearest city or the path which has more pheromone deposits. After each iteration, the pheromone is updated, and the best route is selected based on the shortest distance, time or capacity.

2.3 Critique

The variations and algorithms mentioned in section 2.1 handle most of the time single objective functions which either gives the best distance or estimates the required number of vehicles needed for completing the requests. No research has been done that seeks to maximize delivered goods based on vehicle capacity and minimize distance (or time) travelled. Hence, existing methods cannot provide a complete solution for our problem. Ant Colony Optimization solves the problem very quickly and is flexible to handle dynamic requests.

CHAPTER 3

PROPOSED SOLUTION AND RESULTS

3.1 Problem Specification

We call our system The Pick-up and Drop-off Multiple Travelling Salesman Problem (PD-MTSP). Our goal is to maximize the number of boxes delivered at different locations on a route using vehicles with different capacities while minimizing the distance. Initially, the vehicles that are available to deliver and pick up requests are connected through an application. When the requests that come from customers to pick up boxes crosses a threshold in terms of the number of boxes, one or more vehicles are assigned to satisfy customer requests to pick up and drop off boxes. PD-MTSP then works out the routes and assigns the best vehicle to satisfy the request. We assume that the driver has sufficient time available at his disposal to deliver and pick up the goods or boxes. The driver then travels to the initial pickup location and starts scanning the boxes that need to be delivered to the addresses. At the initial pickup location, the driver does not deliver any goods. The driver then loads the vehicle with the scanned boxes and delivers them to customers following the generated route. Along the route, the driver may pick up boxes for delivery to other locations. Of course, the vehicle must have sufficient capacity or space

pick up requests.

We look at following variations of this scheme:

- Multiple pickups and multiple drop-offs using a single driver.
- Multiple pickups and multiple drop-offs using multiple drivers starting at the same time.
- Multiple pickups and multiple drop-offs using multiple drivers starting at different times.
- Multiple pickups and multiple drop-offs using multiple drivers starting at different times and available for different times.

These variants help to maximize the number of boxes delivered along the route while minimizing the total distance traveled. Handling two objective functions makes our problem a bi-criterion NP-hard problem.

Given *n* salesmen and *m* cities, each salesman starts at an initial location and starts visiting all the cities in his route at least once. The PD-MTSP will generate the route for the driver. The number of cities *m* is always greater than the number of drivers, i.e., *n* (*m*>*n*). In this problem, the drivers do not start from a central location (depot). Instead, they are dispersed around the cities. PD-MTSP automatically identifies the drivers, once they come online and assigns them a route starting at their initial pickup location to handle multiple requests. The vehicle capacities vary based on the type of vehicle. Bigger vehicles can carry more boxes.

3.2 Swarm Intelligence

Ant Colony Optimization belongs to the Swarm intelligence group of algorithms [13]. The main idea of Swarm intelligence is to study the behavior patterns of different social insects like bees, ants, etc., and introduce the same patterns into technology to simulate the process based on their behavior. One such metaheuristic algorithm which follows this swarm intelligence is Ant Colony Optimization. The Ant Colony Optimization algorithm follows the natural behavior of ants that are searching for food laying pheromone tracks as they go. The ants move around in search of food laying down pheromone so that other ants follow the trail as seen in figure 3.1.

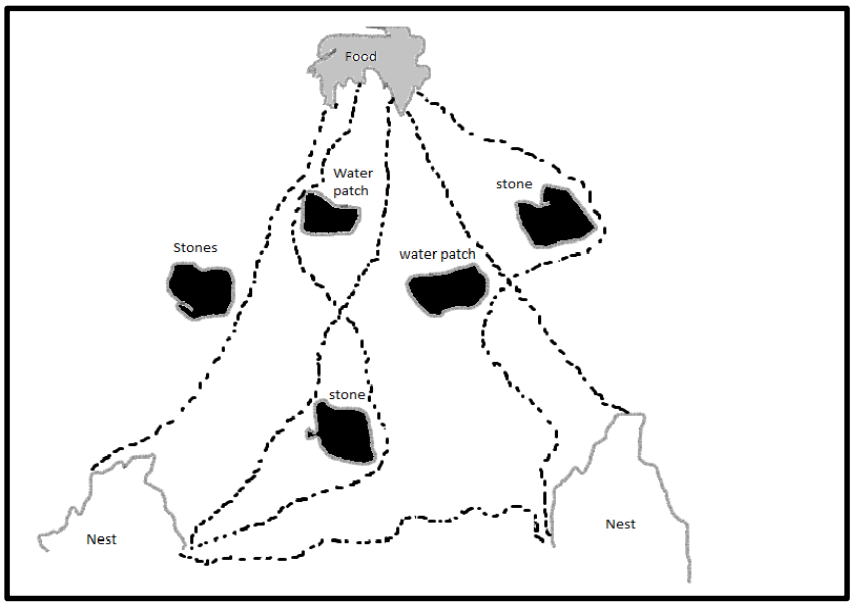

Figure 3.1: Natural behavior of Ant

3.2.1 Ant Colony Optimization

The PD-MTSP problem deals with two parameters. One is the distance (or time) between cities, i.e., the distance between two cities i and j represented as $D_{ij.}$ The second parameter is the capacity C_k of vehicle k.

The Ant Colony Optimization algorithm is an artificial intelligence algorithm that can be applied to combinatorial problems like PD-MTSP where different possible routes are searched for a feasible route with the minimum distance to supply a maximum number of boxes. Ant Colony Optimization(ACO) is a type of search algorithm that seeks the best feasible solution using the pheromone trails of artificial ants. Artificial ants follow the same pattern of behavior as natural ants. Natural ants search for food while laying pheromone on their path, using which other ants follow the same path. Once they reach the food, they head back on the same path to the initial starting location. This traveling of an ant increases the pheromone deposit in the path. Pheromones are not only deposited, but they also evaporate over time. Hence, if a path has not been used for some time, it will contain less pheromone. There may be other ants that follow another path to the same food in a shorter distance that results in more pheromone being deposited than the previous trails. These pheromone deposited paths make other ants follow the new shorter route while slowly evaporating. Artificial ants follow the same procedure while searching for a route that visits all the cities and complete customer requests to pick up and drop off boxes. Each edge between the cities has an initial pheromone level so that no route dominates the other routes. Once a route has been constructed based on the heuristics (shortest distance) using the Ant Colony Optimization algorithm, the edges that belong to the solution are chosen for reducing the pheromone deposits to find other possible routes. The same process of

route construction is repeated for some iterations to find different possible solutions around the initial best. The final output is selected, such that the total route length is the minimal distance while delivering the maximum number of boxes.

3.3 Proposed Ant Colony Optimization Algorithm

The Ant Colony Optimization algorithm initially constructs the route by selecting neighboring edges from the driver's current location, based on probability of pheromone levels and heuristic values, along with the boxes to deliver at nodes, until it meets the ending criteria of the algorithm. Once the route for an iteration is constructed the pheromone on the edges is updated, and the next iteration takes place. The algorithm ends when a feasible route that meets the requirements is obtained.

3.3.1 Selection of Neighboring Nodes

Each salesman starts from the initial pick up location, where he loads the boxes into the vehicle. The ant (driver) selects the next city to be visited from the neighboring nodes, initially using the probability that is calculated based on the pheromone and heuristic values between neighboring nodes. Then the selected node is checked for the deliverables based on the pickup requests. If the selected node is not a delivery point, the next best node that is a delivery point is selected. Once it gets the node based on the probability and the boxes that need to be delivered, this node will be the current node and the construction of the route continues till it reaches the ending criteria.

$$j = \arg\max\{(\tau_{iu})(\eta_{iu})^{\beta}\}\ for\ u\ \notin\ R\ ,\ if\ q\ \leq\ q_0 \qquad (1)$$

otherwise S

where, $\eta_{iu} = \frac{1}{D_{iu}}$

D_{iu} is the distance between the node *i* and its neighbors *u*

τ_{iu} is the amount of pheromone laid on a path between node *i* and neighboring locations *u*. The pheromone laid on the edges of the route between node *i* and neighboring node *u* is initially the same for all the edges. η_{iu} is the inverse of distance between *i* and the neighboring node *u*. β is the weight of the heuristic i.e., selection based on the shortest distance ($\beta > 0$). *R* is the list of nodes that are already visited and stored in the memory. q is a random uniform variable that lies in the range of 0 and 1. q_0 is a parameter.

If $q > q_0$, then the ant selects the next node randomly from unvisited neighboring nodes based on the following probability distribution function,

$$P_{ij} = \frac{(\tau_{ij})(\eta_{ij})^{\beta}}{\sum_{u \notin R} (\tau_{iu})(\eta_{iu})^{\beta}} \text{ if j} \notin \text{R , Otherwise 0} \qquad (2)$$

Based on the above equations, the next neighbor is selected either by the heuristic value or randomly using the probabilistic distribution around the nodes.

3.3.2 Pheromone Update

As mentioned, the ant lays pheromone on the path it travels. The initial pheromone is same for all the edges so that no edges dominate while constructing the route. Once a possible route is constructed using the ACO, the edges in the route will have their pheromone updated using the below formula,

$$\tau_{ij} = (1 - \alpha)\, \tau_{ij} + (\alpha)\, \tau_0 \qquad (3)$$

α is the parameter that controls the speed of evaporation on the edges in the route. τ_0 is the inverse of total length of the individual route. τ_{ij} is the pheromone value between nodes *i* and *j*.

3.3.3 Ending Criteria

The driver will not drop any goods at the initial pickup location, and will not pick up any goods at the last destination in the route. The route is constructed based on the probability of the heuristic values and capacity of the vehicle. Once the node is selected, the vehicle capacity is updated automatically before going to the next node. The route construction is continued till the vehicle does not contain any boxes, or all the nodes have been visited by the vehicle. The total number of boxes that are required to be delivered is denoted by the term $G_{D,}$ and the total amount of boxes that must be picked up is denoted by the term G_P. So the ending criterion is represented as follows, $G_P - G_D = 0$.

3.4 Variations and Results

The following variations have been designed considering different conditions such as drivers starting at the same time, handling multiple pickups and multiple deliveries, and handling single and multiple drivers to achieve the goal mentioned above.

3.4.1 Multiple pickups and multiple deliveries with single driver (MPMD – SD)

The Travelling Salesman problem is a well know combinatorial problem, where each salesman finds the shortest path, to visit, all the cities at least once. In the TSP problem, the salesman starts and ends at the depot. In this variation, i.e., handling multiple

pickups and multiple deliveries with a single driver, there are multiple requests that need to be handled by a single driver, unlike the normal TSP problem. At each node (city/address), there are multiple requests for picking of boxes, which need to be delivered to other nodes (cities/addresses). In this variation, the driver does not start at a depot or any specific location; it can be any place such as a home or any random location in the city. Once the driver is ready to take up the requests, the details are provided to the system such as driver location and vehicle capacity. In this variation, we have assumed that the driver is starting at one of the locations where the requests for picking up boxes are available, instead of a random location. However, in a real-time application, the drivers can start at any location as stated above, and from that location, the nearest pickup location (node) is selected, and the driver can start picking up boxes. The map for the requests is taken in the form of a graph with edges and nodes as shown in figure 3.2, where edges represent the route between two nodes and nodes represent cities or addresses where the requests are available. A request is defined as picking up boxes at a certain location where boxes are available for delivery.

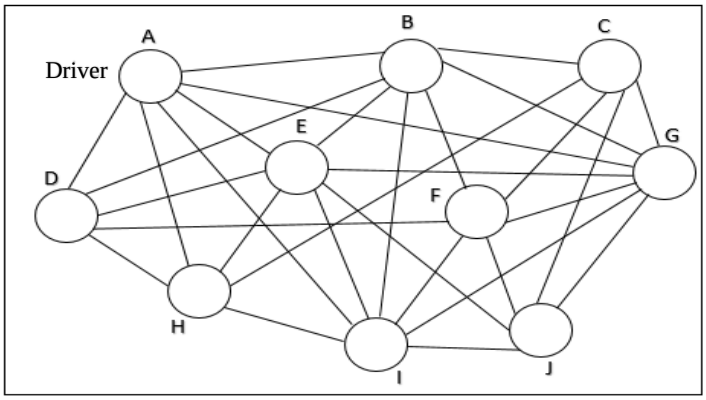

Figure 3.2: Sample graph representing nodes and edges with single driver where each node is a pickup and/or delivery point

As stated above, at each node, there will be multiple requests for picking up boxes and each node can be a pickup location and a delivery point, such as at node *A*, where the driver may have to pick up boxes that need to be delivered at some of the directly connected nodes such as *D, G, H* and *I*. Similarly, the driver may have to pick up boxes at other nodes i.e., *B, C, D,..., J* as shown in figure 3.2 and deliver at different delivery points. At each delivery point, i.e., *D, G, H* and *I*, there may be multiple boxes that need to be delivered, and those boxes are picked up at node *A*. The distance between the nodes, i.e., D_{iu}, where *i* is the current node, and *u* is the set of neighboring nodes is represented in the form of a matrix. The distances in the matrix shown in table 3.1 are distances between 20 cities in Oklahoma obtained from google maps. The cities are listed below in table 3.2.

Table 3.1: Distance matrix for a graph of 20 nodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	64	68	95	63	160	149	116	192	54	172	219	263	183	97	134	330	195	54	33
2	64	0	87	152	107	99	87	112	129	22	109	197	242	161	76	169	326	174	119	33
3	68	87	0	134	115	196	145	82	159	114	165	273	317	246	170	179	274	122	81	70
4	95	152	134	0	46	247	237	212	280	136	259	204	249	174	133	50	394	220	80	142
5	63	107	115	46	0	201	191	185	235	91	213	159	203	123	88	65	388	242	105	90
6	160	99	196	247	201	0	101	209	226	119	50	107	152	145	98	264	424	272	214	127
7	149	87	145	237	191	101	0	129	113	109	54	202	246	240	155	254	331	202	204	118
8	116	112	82	212	185	209	129	0	81	133	181	308	352	271	186	249	227	75	156	101
9	192	129	159	280	235	226	113	81	0	150	167	325	369	288	203	296	227	118	247	160
10	54	22	114	136	91	119	109	133	150	0	132	183	227	146	60	154	347	195	109	42
11	172	109	165	259	213	50	54	181	167	132	0	156	200	194	145	276	374	255	226	140
12	219	197	273	204	159	107	202	308	325	183	156	0	43	84	114	201	523	371	263	224
13	263	242	317	249	203	152	246	352	369	227	200	43	0	109	159	246	567	415	308	268
14	183	161	246	174	123	145	240	271	288	146	194	84	109	0	78	140	486	334	231	187
15	97	76	170	133	88	98	155	186	203	60	145	114	159	78	0	150	401	249	162	102
16	134	169	179	50	65	264	254	249	296	154	276	201	246	140	150	0	452	269	147	152
17	330	326	274	394	388	424	331	227	227	347	374	523	567	486	401	452	0	174	324	316
18	195	174	122	220	242	272	202	75	118	195	255	371	415	334	249	269	174	0	150	164
19	54	119	81	80	105	214	204	156	247	109	226	263	308	231	162	147	324	150	0	88
20	33	33	70	142	90	127	118	101	160	42	140	224	268	187	102	152	316	164	88	0

Table 3.2: Key of cities shown in table 3.1

Nodes	Cities
1	Stillwater
2	Oklahoma City
3	Enid
4	Bartlesville
5	Tulsa
6	Ardmore
7	Lawton
8	Taloga
9	Sayre
10	Harrah
11	Waurika
12	Hugo
13	Idabel
14	Wilburton
15	Holdenville
16	Vinita
17	Boise city
18	Buffalo
19	Newkirk
20	Guthrie

3.4.1.1. Implementation of MPMD – SD

MPMD – SD, i.e., Multiple Pickup and Multiple Drop-off with Single Driver described above was implemented using java. The following assumptions are made in implementing MPMD - SD,

- The driver starts at one of the nodes, where there is a request to pick up boxes
- The driver has no time restrictions
- The requests are known prior to the drivers starting from the first point.

Requests at each node are randomly generated, i.e., at each node a random number of nodes are selected as delivery points, and at each node, a random number of boxes are randomly generated. These requests are generated based on the number of nodes present in the graph. The distance between nodes is taken from the distance matrix, where the

distances between the nodes are randomly generated. Once the information regarding the map is generated by the system, driver information is taken as input, i.e., driver's starting location and vehicle capacity. Based on this information, the modified ant algorithm runs through the map (generated as mentioned above) and generates the route with the shortest distance while delivering a maximum number of boxes.

Algorithm

The ant algorithm is modified as below to optimize the number of boxes picked up and delivered by the vehicle. The next node is selected using the above probability equations, but it may lead to a node where the driver has no goods to deliver. Therefore, the node that is generated by the ant colony algorithm is always checked for the deliverables. If there is a delivery, then the node is added to the route, if not, the next best nodes are selected based on the heuristics. By this, at every node, the driver delivers the goods and can pick up new requests, which increases the number of boxes handled. Also, at every node, before delivering boxes, all the visited nodes are checked for the picked up boxes to deliver at this node. All the boxes picked up to deliver at this node so far are summed up and delivered at this location, and new boxes picked up if any. The driver is considered as an ant and the number of ants are equal to the number of drivers present in the system. In the algorithm, when there are no neighboring nodes, that have delivery requests from the current node, the system iterates through the visited nodes and pickup requests handled by the driver (ant) so far and finds the node that has the maximum number of boxes that need to be delivered. The selected node is taken as *maxnode* and the number of boxes that need to be delivered are taken as *maxval* as shown below.

```
Initialize the parameters and pheromone
driver.current_node ← drivers_start_location
//assign nodes to driver's route until all requests are completed
while requests_list.isEmpty() do
    selected_node ← generate_next_node (driver.current_node, driver)
    //adds the non-visited nodes to the list
    def. generate_next_node (driver.current_node, driver)
        for each node i ∈ neighboring_nodes do
            if !visited.contains[i] then
                non_visited_neighbors ← i
            if non_visited_neighbors.isEmpty() then
                for each node i ∈ visited do
                    temp ← deliveries(i)
                for each node j ∈ neighboring_nodes do
                    if temp.contains(j) then
                        if delivery_track.get(j)! = 1 then
                            val ← deliveries.get(j)
                            neighbor_list_deliveries ← j
                for each k ∈ neighbor_list_deliveries do
                    if temp.contains(k) then
                        if maxval < val then
                            maxval ← val
                            maxnode ← k
                if maxnode == 0 then
                    maxnode ← get_random(neighboring_node.get(current_node))
                    selected_node ← maxnode
                    vehicle_loading (selected_node, driver)
                    while vehicle_overloaded == 1 do
                        neighboring_list.remove(selected_node)
                        selected_node ← get_random(neighboring_node.get(current_node)
                        if neighboring_list.isEmpty() then
                            pending_queue ← selected_node
                            vehicle_overloaded ← 0
                            vehicle_loading (selected_node, driver)
                else maxnode != 0 then
                    vehicle_unloading (maxnode, driver)
                    vehicle_loading (maxnode, driver)
                    if vehicle_overloaded == 1then
                        pending_queue ← maxnode
                        vehicle_overloaded ← 0

            else !non_visited_neighbors.isEmpty()
                selected_node ← probability (current_node, neighboring_nodes)
                temp_list ← deliveries.get(current_node)
                if temp_list.contains(selected_node) then
                    vehicle_unloading(selected_node, driver)
                    vehicle_loading(selected_node, driver)
                    if vehicle_overloaded == 1then
                        pending_queue ← maxnode
                        vehicle_overloaded ← 0
                else ! temp_list.contains(selected_node) then
                    if non_visited_neighbors.size > 0 then
                        tempvisit ← selected_node
                        selected_node ← generate_next_node (driver.current_node, driver)
```

Figure 3.3: Modified ant colony algorithm

The *vehicle_loading* function as shown in figure 3.4, checks the capacity of the vehicle and finds the number of boxes to be picked at the next node. If the total number of boxes exceeds the vehicle limit of handling boxes, the variable *overloaded* is set to 1. When *overloaded* is set 1, the next best neighbor is selected and checked for the capacity constraint. If none of the available nodes satisfies the criteria, then the selected node is sent to the pending queue, and the pickup request is not serviced and, only boxes will be delivered at that point.

```
def vehicle_loading (current_node, driver)
        if pickup_track == 1 then
          get_deliveries ← deliveries(current_node)
        for each i ∈ get_deliveries do
          val ← val + get_deliveries.get(i)
  if (driver.vehicle_storage + val) > driver.vehicle_capacity then
        vehicle_overloaded = 1;
              else (driver.vehicle_storage + val) < driver.vehicle_capacity then
               driver.vehicle_storage ← driver.vehicle_storage + val
        pickup_track ← 0
```

Figure 3.4: Vehicle loading function

The *vehicle_unloading* function checks all the visited nodes and tracks the boxes that have been picked up so far to be delivered to the selected node. This step helps to reduce revisits of nodes and delivers all the boxes once at every node. Handling all boxes to deliver once at each node, makes space to handle further requests.

```
def vehicle_unloading(current_node, driver)
        for each i ∈ visited do
            temp_deliveries ← deliveries.get(i)
        if temp_deliveries,contain(current_node) then
            if deliver_track != 1then
              val ← val + temp_deliveries.get(current_node)
          driver.vehicle_storage = driver.vehicle_storage – val
```

Figure 3.5: Vehicle unloading function

Based on the above-modified ant algorithm, the ant selects the next node either based on shortest distance or based on the boxes it needs to deliver. The following parameters are used in the implementation of above algorithm for MPMD – SD [13] [15],

$q \leftarrow$ random variable between 0 and 1, $q_0 \leftarrow 0.9$, $\alpha \leftarrow 0.01$, $\beta \leftarrow 4$. The best results are obtained at these values after implementing with several different values. The initial pheromone value is set to the smallest value greater than 0; in our case we have taken 0.8 as initial pheromone value. By using the above parameters, we achieved the shortest distance while maximizing the number of boxes delivered compared to existing algorithms like nearest neighbor algorithm.

3.4.1.2 Results for MPMD – SD

The table below shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above.

Table 3.3: Input for MPMD-SD simulation

Driver	Starting City	Vehicle Capacity
D1	1	500

The below table 3.4 shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.4: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	86	61
2	21	54
3	64	79
4	57	37
5	68	53
6	59	96
7	56	42
8	91	92
9	91	79
10	91	20
11	69	110
12	70	59
13	48	62
14	42	61
15	54	93
16	65	83
17	75	44
18	81	66
19	19	43
20	108	81

Below are the results obtained for a single driver visiting 20 different cities with 500 boxes capacity limit. The number of boxes delivered with respect to capacity and total distance traveled is shown in figure 3.6.

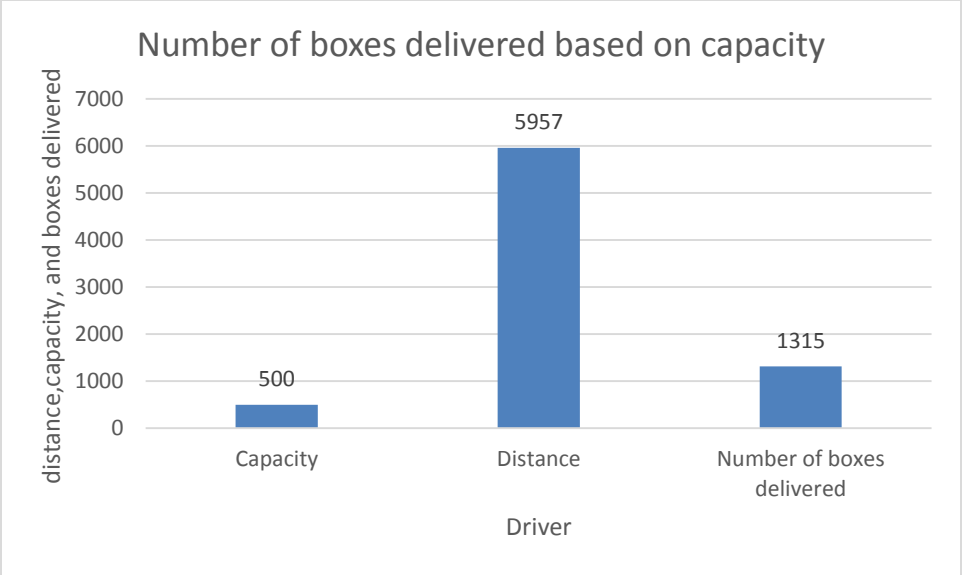

Figure 3.6: Number of boxes delivered by single driver

3.4.2 Multiple pickups and multiple deliveries with multiple drivers starting at the same time (MPMD – MD)

In this variation, all the available salesmen start at different locations. There is no particular depot in our variation, unlike normal MTSP problems. The vehicles are connected through the application, and all the available drivers are assigned with nearby requests as stated above. The graph considered in this variation is a symmetric graph as shown in figure 3.7, and represented through nodes and edges. Each node had multiple requests for picking up the boxes that need to be delivered to other nodes.

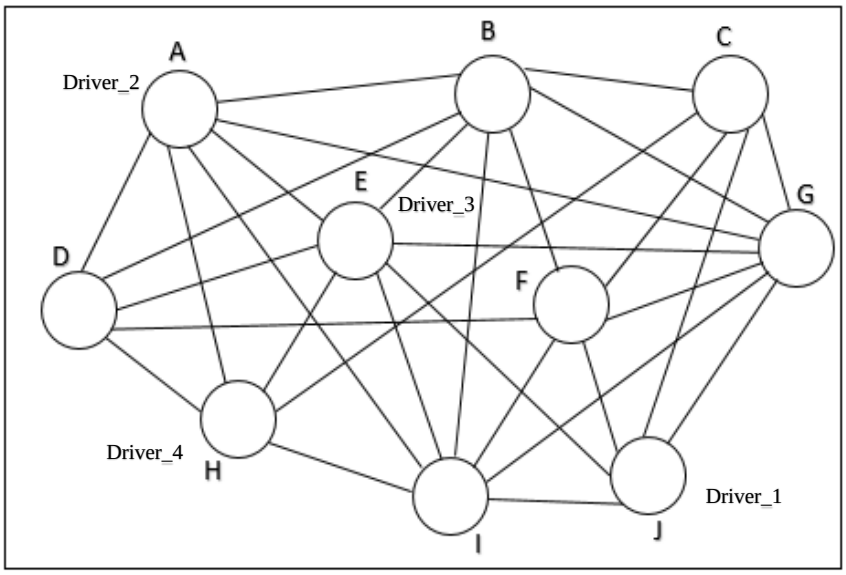

Figure 3.7: Sample graph representing nodes and edges with multiple drivers

The requests are generated randomly as stated before and the distance between the nodes, i.e., D_{iu}, where i is the current node and u is the set of neighboring nodes is represented in the form of a matrix. The distances in the matrix, as mentioned are taken from google maps considering 20 different cities from Oklahoma state as shown in table 3.1, and the distance the between i to j is equal to the distance between j to i.

3.4.2.1 Implementation of MPMD - MD

MPMD – MD, Multiple pickups and multiple deliveries with multiple drivers is implemented using java. The following assumptions are made while implementing MPMD – MD:

- Drivers start at one of the request nodes
- Drivers have no time restrictions

- Limited number of requests are available
- All drivers start at the same time for processing requests

In this variation, we know the number of drivers that are available to take up the requests upfront. Multiple requests are generated at each node randomly based on the number of nodes present in the graph. The distances at each edge between nodes are taken from the distance matrix as stated above. Each driver has a different starting location and varied vehicle capacity. The input of the algorithm in this variation is the driver's starting location and vehicle capacity along with the number of drivers available. The modified ant algorithm runs through the graph with multiple requests and assigns the route to each driver. The distribution of the nodes is equal among the multiple drivers based on the FCFS (First Come First Serve) basis. No two drivers try to pick the boxes at the same location in this variation. He or she might visit the location to drop-off the boxes, but no pickup request is carried out if it is assigned to some other driver.

Algorithm

The ant colony algorithm is modified as above (figure 3.3) to handle MPMD – MD. The main issue that comes while handling MPMD – MD is tracking the boxes picked up by drivers. Drivers cannot handle requests which are already assigned to some other drivers. Hence a tracking system is needed to keep track of the deliveries. All the deliveries are tracked by the *deliveries_track* set where initially all the deliveries are assigned 0, and when it is picked by a certain driver, the node is added to the *driver_picked_requests*, and the value is changed to 1. Therefore, other drivers can have pickups only if the *deliveries_track* is set to 0. In this way, no two drivers can go for the same pickup request. In the same way, while delivering the boxes, the system updates the vehicle storage by

removing the boxes which driver has picked up. Once the vehicle storage is updated, the value is set to 2, which means the request has been completed. So while assigning nodes, the modified ant algorithm will not include these nodes in assigning them to a driver, since they have already been processed by one of the drivers.

```
for each driver D_i ∈ D_n do
    D_i ← driver starting location
    C_i ← vehicle capacity
    Visited[i] ← starting location
    Route[i] ← starting location
while pickup_track.isEmpty() && deliver_track.isEmpty() do
  for each driver D_i ∈ D_n do
      selected_node ← generate_next_node (driver.current_node, driver)
      driver.current_node = selected_node
      Route[i] ← selected_node
      Visited[i]  ← selected_node
```

Figure 3.8: Algorithm for MPMD - MD

The *generate_next_node* function is similar to the algorithm shown in figure 3.3. Nodes are assigned to the driver until all the available requests are processed, and picked boxes are delivered by all the drivers. The *vehicle_loading* and *vehicle_unloading* functions work similarly to what is shown in figures 3.4 and 3.5 respectively. The following parameters are used in the implementation of the above algorithm for MPMD – MD, $q \leftarrow$ random variable between 0 and 1, $q_0 \leftarrow 0.9$, $\alpha \leftarrow 0.01$, $\beta \leftarrow 4$. Multiple values are considered for β such as 2.3, 3, 4, and 8 [13] [15], and the best results are seen at $\beta \leftarrow 4$. Similarly, α values are also tried with 0.1, 0.001, and 0.01 and the best results are achieved with the above parameter value. The shortest distance is compared to the results obtained using brute force and nearest neighbor algorithms.

3.4.2.2 Results for MPMD – MD

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above. The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.5: Input for MPMD-MD simulation

Driver	Starting City	Vehicle Capacity
D1	1	100
D2	8	300
D3	18	500

Table 3.6: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	21	79
2	62	53
3	47	63
4	52	65
5	26	50
6	32	45
7	50	51
8	47	49
9	64	68
10	47	52
11	67	35
12	55	50
13	106	17
14	78	61
15	60	60
16	98	55
17	29	59
18	36	40
19	53	64
20	44	58

The below figures and tables shows the results of MPMD – MD, i.e., Multiple Pickup and Multiple Deliveries with Multiple Drivers. These results are based on two scenarios, same capacities, and varied capacities. The distance, number of delivered and picked requests are compared to each driver's vehicle capacity. We achieve the best results as below by using the modified ant colony algorithm.

1) *Varied capacities* with multiple drivers

Table 3.7: Comparison of distance with vehicle capacity

Drivers	Vehicle Capacity	Distance
D1	100	2244
D2	300	3481
D3	500	3447

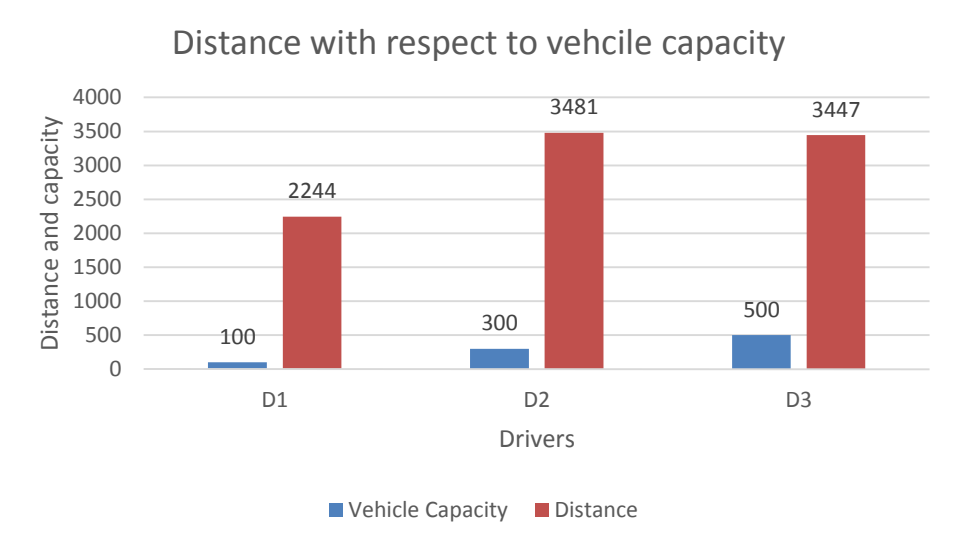

Figure 3.9: Distance traveled by each driver

Table 3.8: Comparison of number of picked requests with the vehicle's capacity

Drivers	Vehicle Capacity	Picked Requests
D1	100	4
D2	300	7
D3	500	9

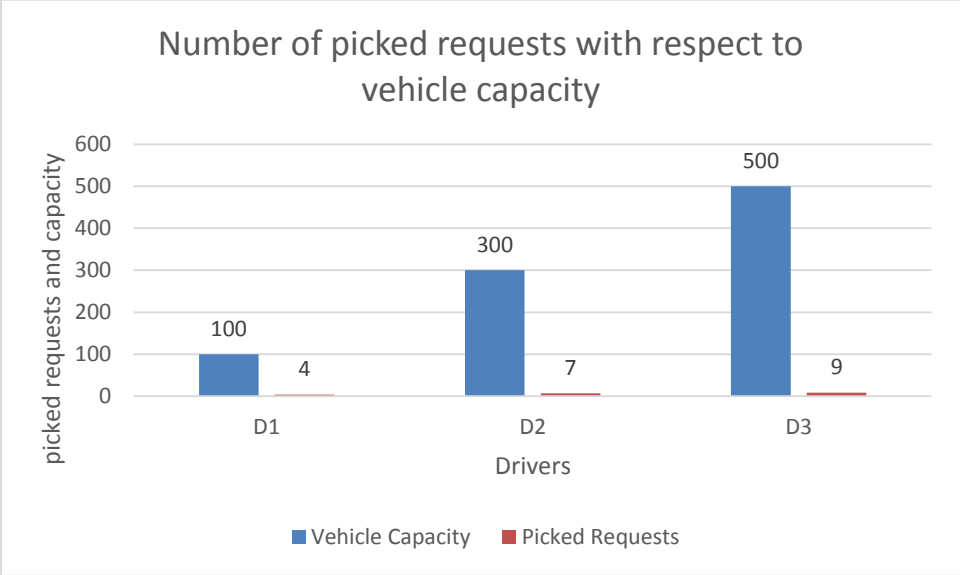

Figure 3.10: Number of pickup requests handled by each driver

Table 3.9: Comparison of number of boxes delivered with the vehicle's capacity

Drivers	Vehicle Capacity	Delivered boxes
D1	100	132
D2	300	374
D3	500	568

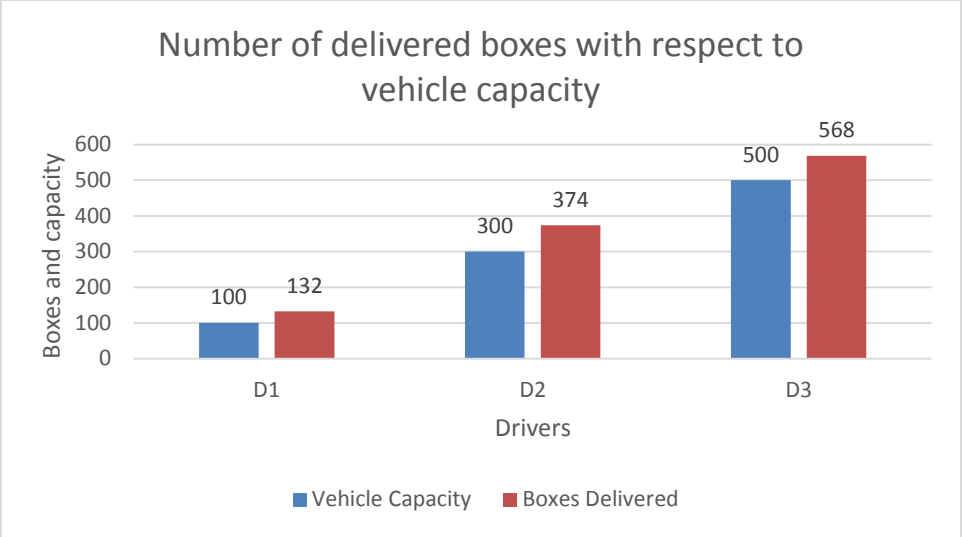

Figure 3.11: Number of boxes delivered by each driver

Table 3.10: Comparison of number of boxes delivered with the total distance traveled and vehicle's capacity

Drivers	Vehicle Capacity	Delivered boxes	Distance
D1	100	132	2244
D2	300	374	3481
D3	500	568	3447

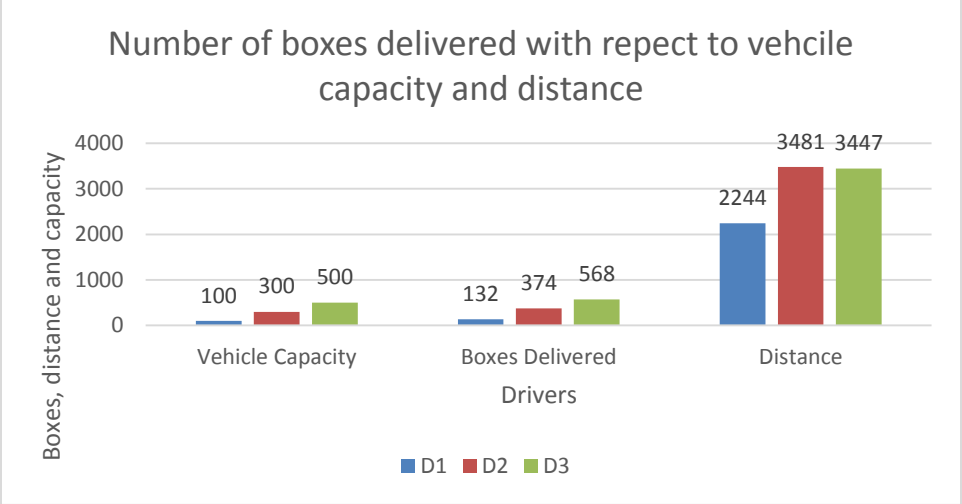

Figure 3.12: Number of boxes delivered by each driver with respect to capacity and total distance traveled

In the above figure 3.12, driver *D3* travelling more distance than the drivers *D1* and *D2*, because of the varied capacities. *D1* can hold only 100 boxes and therefore handles fewer pickup requests than the vehicles with larger capacities i.e., 300 and 500. Similarly, *D2* handles more pickup requests than *D1* and fewer number of requests than *D3* because it can handle more requests compared to *D1* and fewer number of requests compared to *D3*. Hence, the number of delivery points decreases respectively based on the number of pickup requests. Therefore, the driver handling fewer pickup requests travels less distance compared to the driver handling more pickup requests because of capacity constraints.

2) *Same vehicle capacities* with multiple drivers

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above.

Table 3.11: Input for MPMD-MD simulation

Driver	Starting City	Vehicle Capacity
D1	1	300
D2	8	300
D3	18	300

The below table 3.12 shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.12: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	85	53
2	96	52
3	77	73
4	78	88
5	85	67
6	78	79
7	67	80
8	14	60
9	65	69
10	54	82
11	18	48
12	82	52
13	81	41
14	63	93
15	23	48
16	65	76
17	61	106
18	93	78
19	71	51
20	99	59

Table 3.13: Comparison of distance traveled with vehicle's capacity

Drivers	Vehicle Capacity	Distance
D1	300	3317
D2	300	3923
D3	300	4278

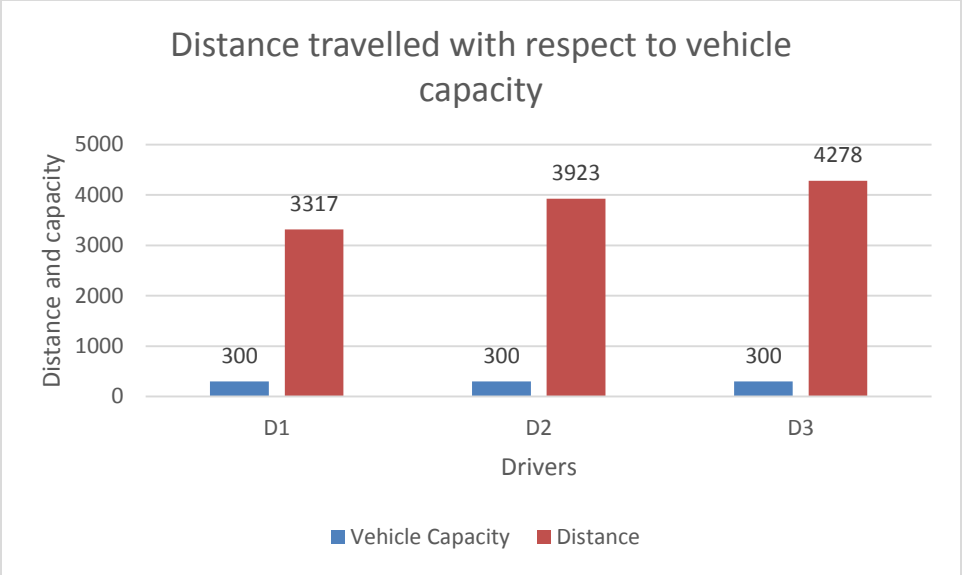

Figure 3.13: Distance traveled by each driver

Table 3.14: Comparison of number of picked requests with the vehicle's capacity

Drivers	Vehicle Capacity	Picked requests
D1	300	5
D2	300	8
D3	300	7

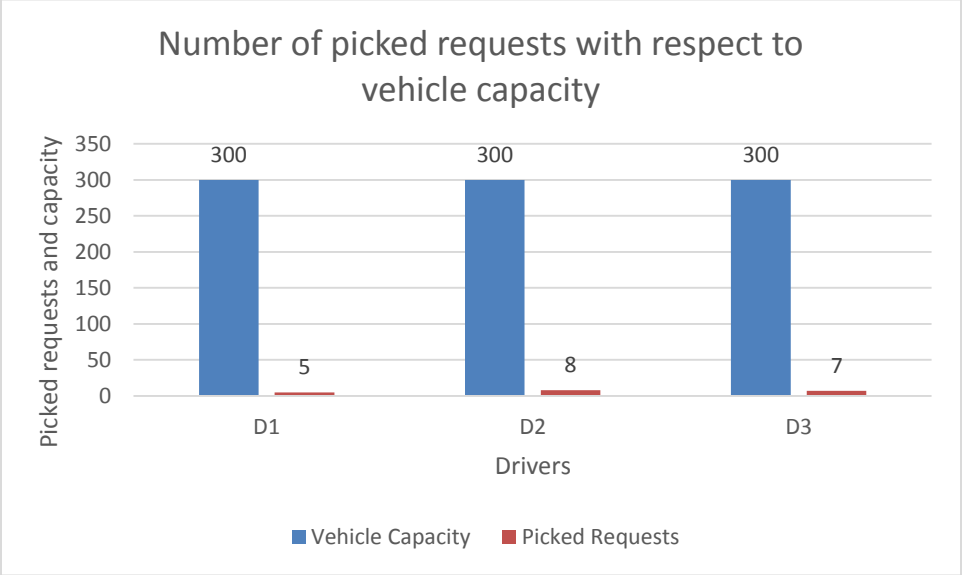

Figure 3.14: Number of pickup requests handled by each driver

Table 3.15: Comparison of number of boxes delivered with the vehicle's capacity

Drivers	Vehicle Capacity	Boxes delivered
D1	300	406
D2	300	439
D3	300	510

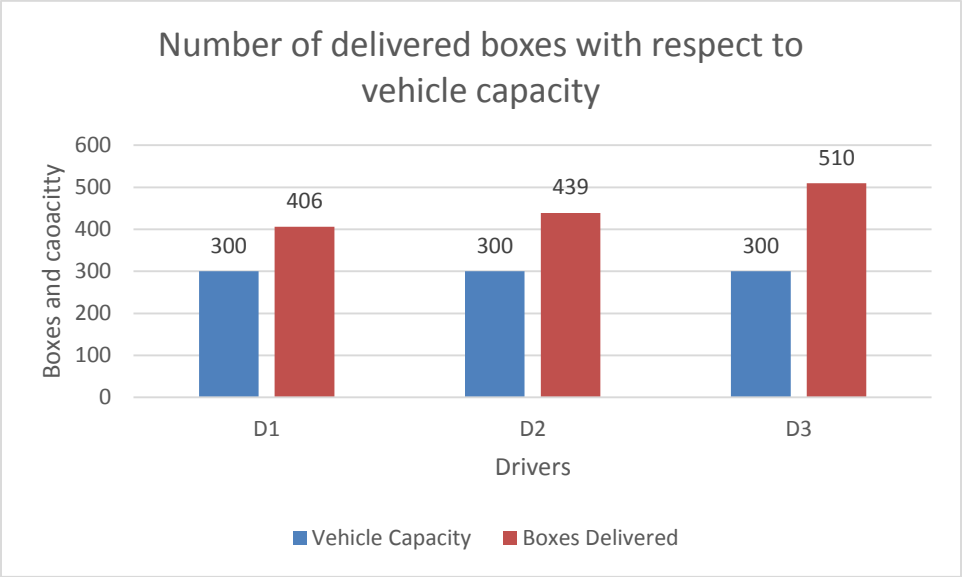

Figure 3.15: Number of boxes delivered by each driver

Table 3.16: Comparison of number of boxes delivered with the total distance traveled and vehicle's capacity

Drivers	Vehicle Capacity	Boxes delivered	Distance
D1	300	406	3317
D2	300	439	3923
D3	300	510	4278

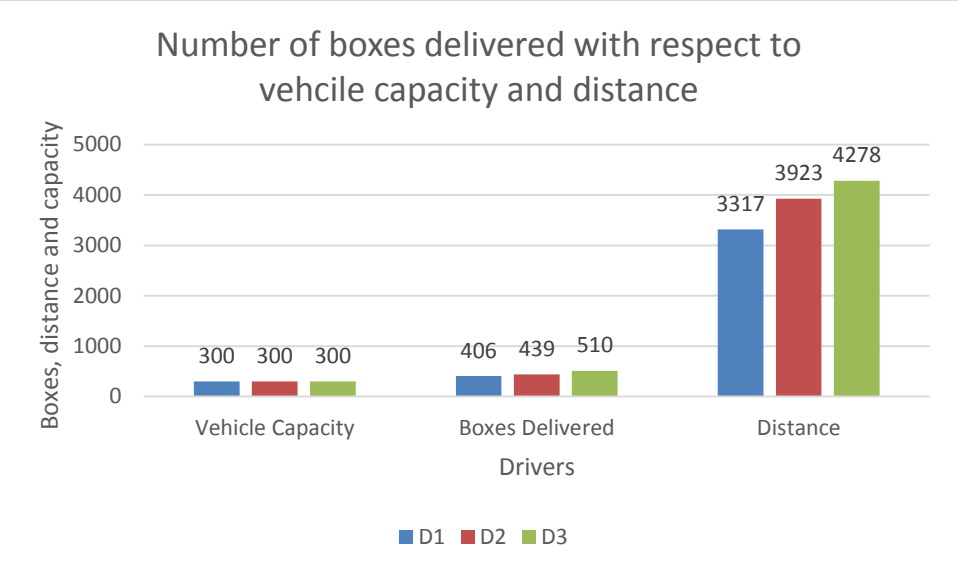

Figure 3.16: Number of boxes delivered by each driver with respect to the total distance traveled and vehicle's capacity

In the above figure 3.16, the vehicles are considered with the same capacities. However, the requests are generated randomly as stated in the implementation section. The distance travelled by the vehicle is proportional to the total number of boxes picked up by the driver. The nodes are assigned to the driver based on a first come first serve basis. The driver can pick up requests only if they are not assigned to the other drivers.

3.4.3 Multiple pickups and multiple deliveries with multiple drivers starting at different times (MPMD – MD_DT)

In MPMD - MD_DT, the drivers start at different locations, and at different times with varied vehicle capacities. They can start at any location, and based on their location; the drivers are given the nearest initial pickup location. The drivers travel to the initial pickup location and pick up boxes for delivery. In this variation, we assume the driver start at initial pickup location. The main issue in handling drivers starting at different times is

to maintain the modified ant algorithm running continuously so that when the drivers come in the algorithm starts assigning the nodes to the drivers based on their vehicle capacity and the nodes that have not been serviced yet. The system takes the driver information once the driver comes online, and start generating the route based on the driver's location. In MPMD - MD_DT, the handling of boxes follows the same rules of MPMD – MD, i.e., when a pickup request is assigned to a driver at a certain location, other drivers will not be assigned the same pickup request, although they can deliver boxes at that node.

3.4.3.1 Implementation of MPMD – MD_DT

MPMD – MD_DT, i.e., Multiple Pickup and Multiple Deliveries with Multiple Drivers starting at different times is implemented using java client-server/ socket programming and java multi-threading. The following assumptions are made for MPMD – MD_DT:

- Driver start at one of the nodes, where there is a request to pick up boxes
- Driver has no time restrictions
- Requests are available prior to the drivers check in.

In MPMD – MD_DT, as stated above, the modified ant algorithm has to run continuously, to process the pickup requests with drivers starting at different times. Hence we used the client-server model for this implementation. The diver information, i.e., driver's location and vehicle capacity are taken as input from the client side and sent to the server side through sockets. Once the driver information is entered, the server responds to the information and the algorithm thread starts running. The algorithm thread starts assigning nodes to the driver based on his or her location and vehicle capacity. When a

new driver comes online, and the driver's information is recorded on the client side and the new driver's information is sent to the server. The thread which is already running with a single driver is updated with the new information and starts assigning nodes to both the drivers with the pickup requests left. The remaining requests are distributed equally among the drivers. In the same way, the algorithm runs continuously until all requests are assigned to the drivers based on their time of entry.

Algorithm

The algorithm for this implementation is divided into two parts; one is the client side, and the other is on the server side. As mentioned above, at the client side, driver information is taken as input, and at the server side, the input is processed. The client-side implementation is shown in the below algorithm in figure 3.17.

```
create the socket object for connecting to a port
while terminate != true do
    driver.starting_location ← starting_location
    driver.vehicle_capacity ← capacity
    visited[i] ← starting_location
    route[i] ← starting_location
    //sends the driver information to server side
    output_stream ← driver_object
    if drivers_available == true then
        continue
    else if drivers_available == false then
        terminate ← true
```

Figure 3.17: Client environment for MPMD-MD_DT

The server-side implementation is shown in below figure 3.18. At the server side, the algorithm is implemented in a separate thread, which executes in parallel to the main

thread. At the main thread, information is taken from the output stream. Once the main thread reads the information from the client side, it updates the algorithm thread.

```
Create the socket connection for clients to get connected
    Thread t
        while true do
            Modified ant algorithm as described in figure 4
    Thread main
        while trip != true do
            if driver_object_client != null then
                driver_object_server ← driver_object_client
                drivers_list ← driver_object_server
                t. start()
            else if driver_object_client == null then
                trip ← true
```

Figure 3.18: Server environment for MPMD-MD_DT

The modified ant algorithm of figure 3.18 works similar to the algorithm of figure 3.3. The *vehicle_loading* and *vehicle_unloading* functions work similar to the algorithms of figures 3.4 and 3.5 respectively. The following parameters are used in the implementation of the above algorithm for MPMD – MD_DT, $q \leftarrow$ random variable between 0 and 1, $q_0 \leftarrow 0.9$, $\alpha \leftarrow 0.01$, $\beta \leftarrow 4$. These values are based on the research in [13][15].

In this implementation, at the server side the system checks for the availability of new drivers after assigning each node to existing drivers. In real time implementation, the requests are added dynamically to the system and hence the system assigns the route to the driver based on his vehicle capacity and availability time. In our simulation, we have considered limited number of cities and hence we increased the time that the system waits to check the driver availability and assigns the nodes once the driver is available to take the requests based on the vehicle capacity and availability time. We implemented this in java by using multi-threading and client-server architecture.

3.4.3.2 Results for MPMD – MD_DT

The below figures are the results visualized in tables and graphs. The results for MPMD – MD_DT are shown for three scenarios, i.e., drivers coming at different time intervals i.e., small time gap, mixed time interval and large time interval. The total distance traveled by each driver, the number of boxes handled by each driver are compared based on vehicle capacity as shown below.

Scenario 1: Drivers entering in short gap of time in intervals of 3 minutes

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above.

Table 3.17: Input for MPMD-MD_DT simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time
D1	1	100	0
D2	8	300	3
D3	18	500	6

The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.18: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	29	55
2	30	40
3	58	65
4	63	43
5	65	72
6	37	45
7	11	44
8	18	64
9	42	38
10	71	26
11	48	40
12	47	63
13	42	34
14	37	53
15	14	74
16	67	56
17	112	51
18	54	9
19	70	35
20	37	45

In this scenario each driver enters at intervals of 3 minutes into the system and once the drivers are checked in to the system, the drivers are assigned their routes. The simulation results are shown below.

Table 3.19: Comparison of total distance traveled with vehicle's capacity

Drivers	Vehicle Capacity	Distance	Driver Arrival Time
D1	100	2094	0
D2	300	2380	3
D3	500	2033	6

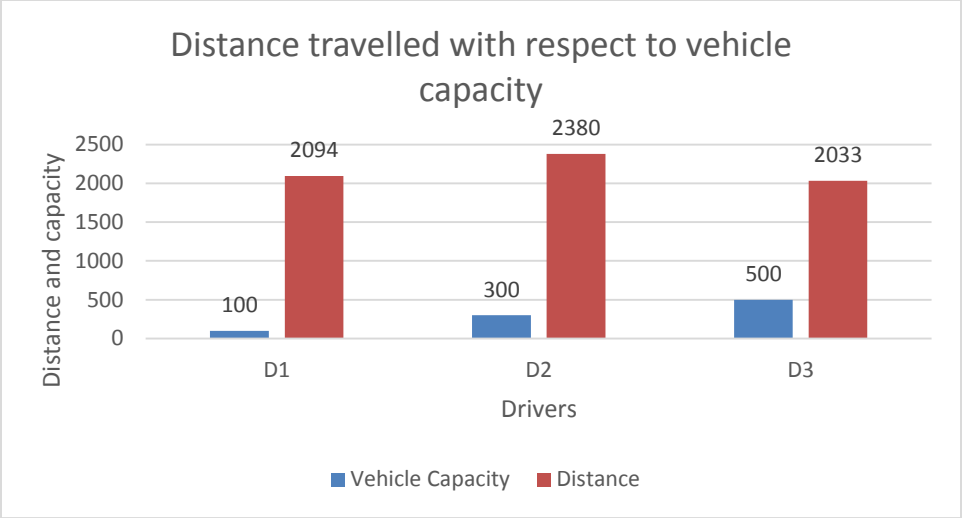

Figure 3.19: Total distance traveled by each driver

Table 3.20: Comparison of number of pickup requests handled with respect to vehicle's capacity

Drivers	Vehicle Capacity	Picked requests	Driver Arrival Time
D1	100	2	0
D2	300	8	3
D3	500	10	6

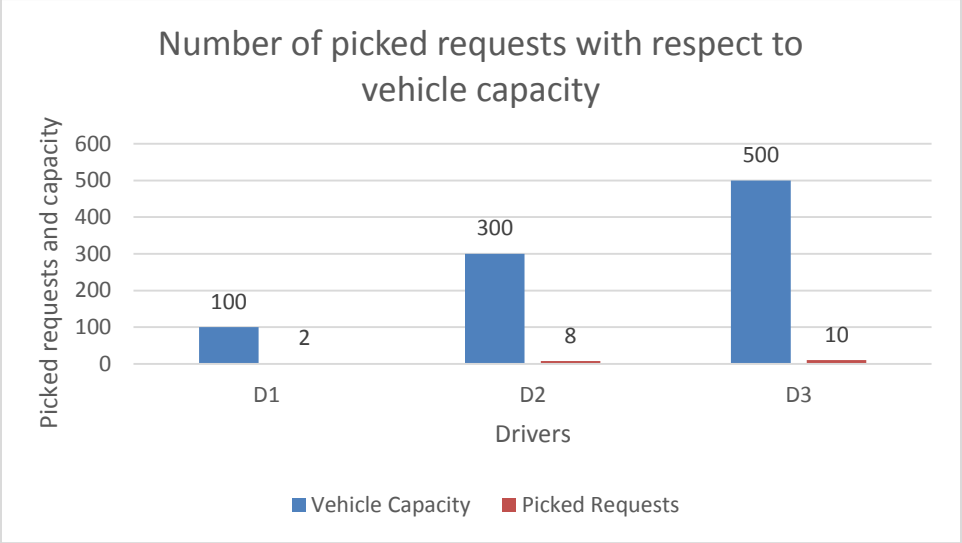

Figure 3.20: Number of pickup requests handled by each driver

Table 3.21: Comparison of number of boxes delivered with respect to the vehicle's capacity

Drivers	Vehicle Capacity	Delivered boxes	Driver Arrival Time
D1	100	87	0
D2	300	404	3
D3	500	461	6

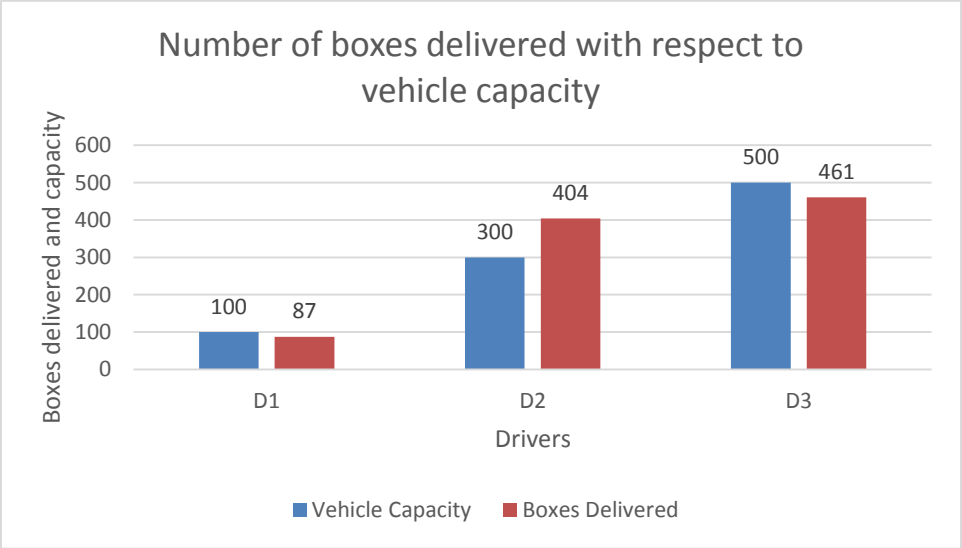

Figure 3.21: Number of boxes delivered by each driver

Table 3.22: Comparison of number of boxes delivered with respect to total distance traveled and vehicle's capacity

Drivers	Vehicle Capacity	Delivered boxes	Distance	Driver Arrival Time
D1	100	87	2094	0
D2	300	404	2380	3
D3	500	461	2033	6

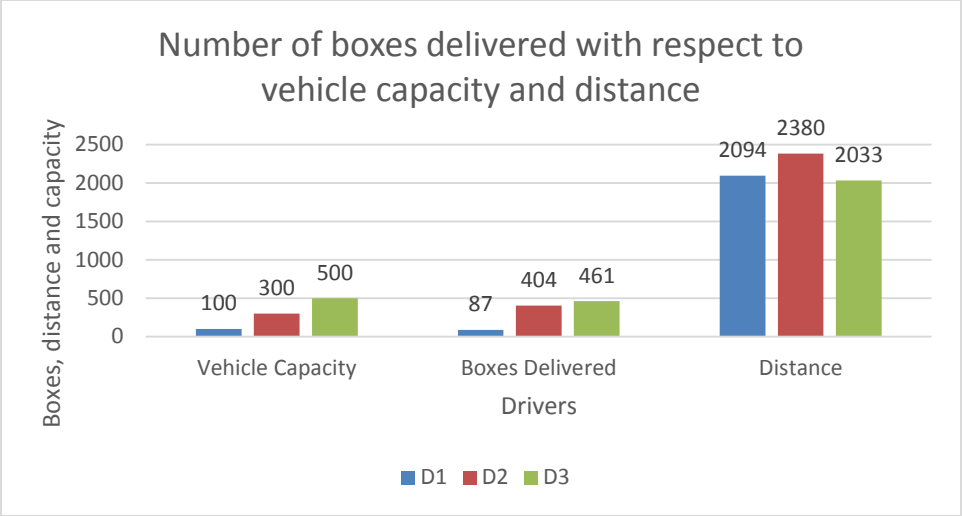

Figure 3.22: Number of boxes delivered by each driver with respect to total distance traveled and vehicle's capacity

Table 3.23: Number of boxes delivered based on the driver's entry time

Drivers	time (min)	boxes delivered
D1	0	87
D2	3	404
D3	6	461

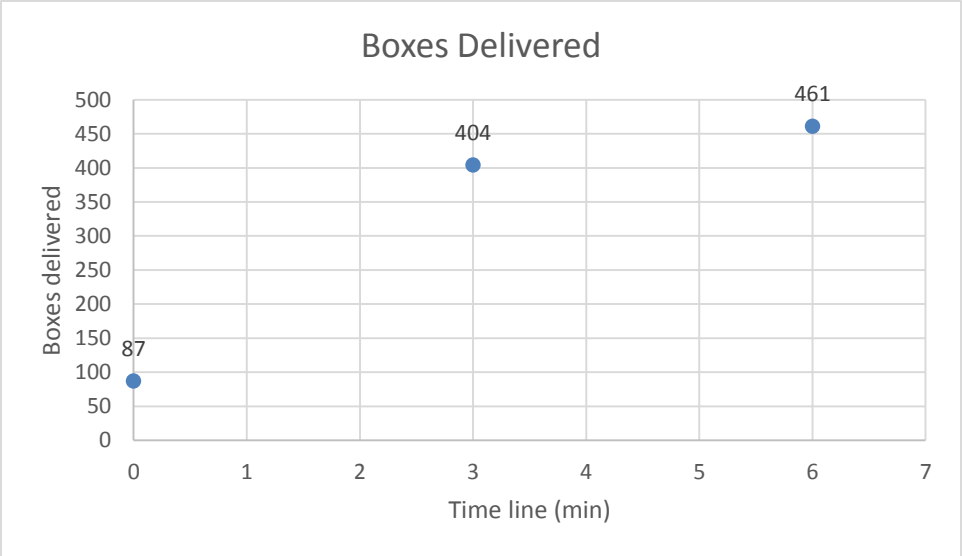

Figure 3.23: Number of boxes delivered based on the entry time of driver's

Scenario 2: Drivers entering in mixed time intervals of 20 and 40 minutes

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above. The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.24: Input for MPMD-MD_DT simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time
D1	1	100	0
D2	8	300	20
D3	18	500	40

Table 3.25: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	17	51
2	61	36
3	51	42
4	47	53
5	70	90
6	75	71
7	16	44
8	68	36
9	55	36
10	38	50
11	80	19
12	37	53
13	25	42
14	34	45
15	93	36
16	50	56
17	51	57
18	20	39
19	24	31
20	48	73

In this scenario each driver enters at mixed interval of 20 and 40 minutes into the system and once the drivers are checked in to the system, the drivers are assigned their routes. The simulation results are shown below.

Table 3.26: Comparison of total distance traveled with respect to vehicle's capacity

Drivers	vehicle capacity	Distance	Driver Arrival Time
D1	100	1761	0
D2	300	1672	20
D3	500	1381	40

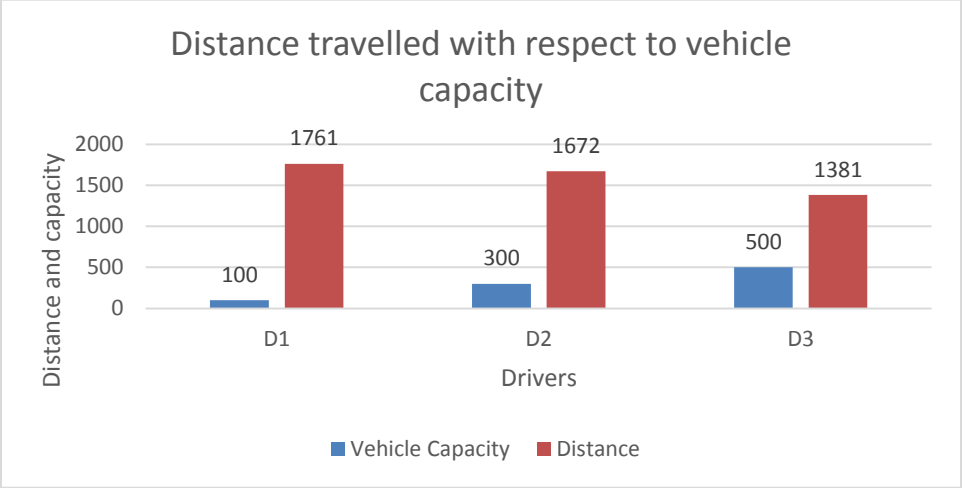

Figure 3.24: Total distance traveled by each driver

Table 3.27: Comparison of number of pickup requests handled with respect to vehicle's capacity

Drivers	vehicle capacity	picked requests	Driver Arrival Time
D1	100	2	0
D2	300	8	20
D3	500	10	40

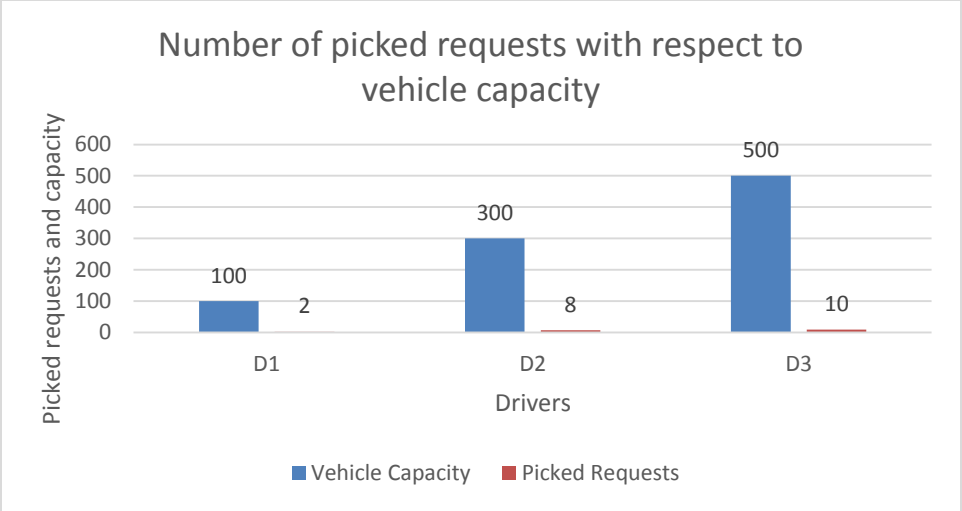

Figure 3.25: Number of pickup requests handled by each driver

Table 3.28: Comparison of number of boxes delivered with respect to the vehicle's capacity

Drivers	vehicle capacity	Boxes delivered	Driver Arrival Time
D1	100	67	0
D2	300	360	20
D3	500	533	40

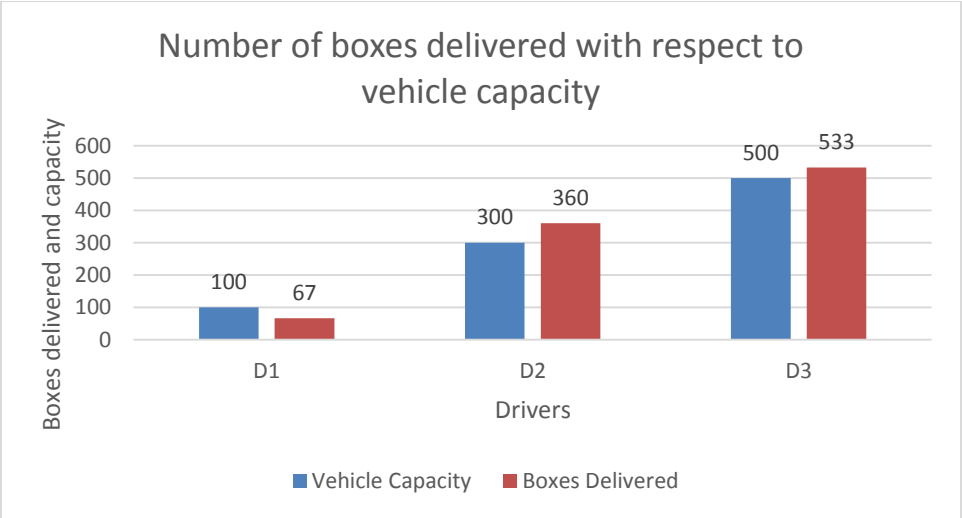

Figure 3.26: Number of boxes delivered by each driver

Table 3.29: Comparison of number of boxes delivered with respect to total distance traveled and vehicle's capacity

Drivers	vehicle capacity	Boxes delivered	Distance	Driver Arrival Time
D1	100	67	1761	0
D2	300	360	1672	20
D3	500	533	1381	40

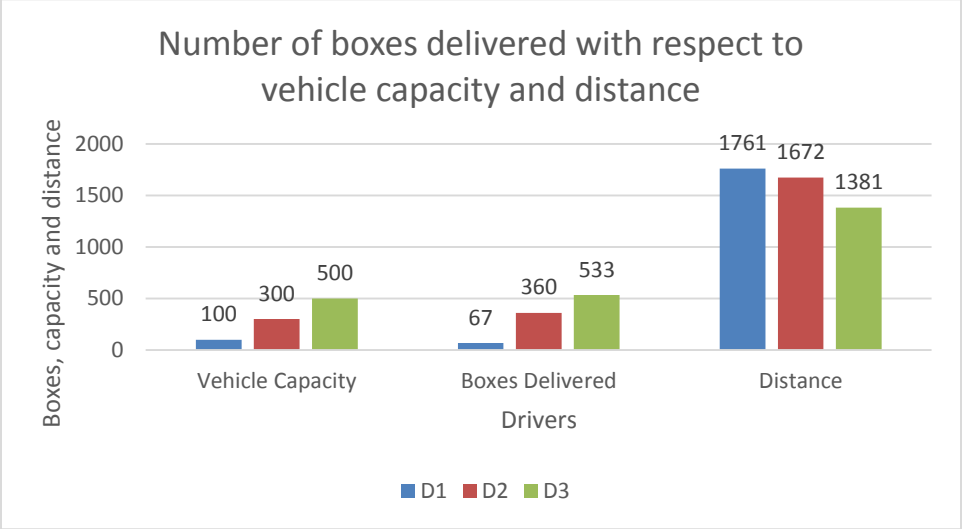

Figure 3.27: Number of boxes delivered by each driver with respect to total distance traveled and vehicle's capacity

Table 3.30: Number of boxes delivered based on the driver's entry time

Drivers	time(min)	Boxes delivered
D1	0	67
D2	20	360
D3	40	533

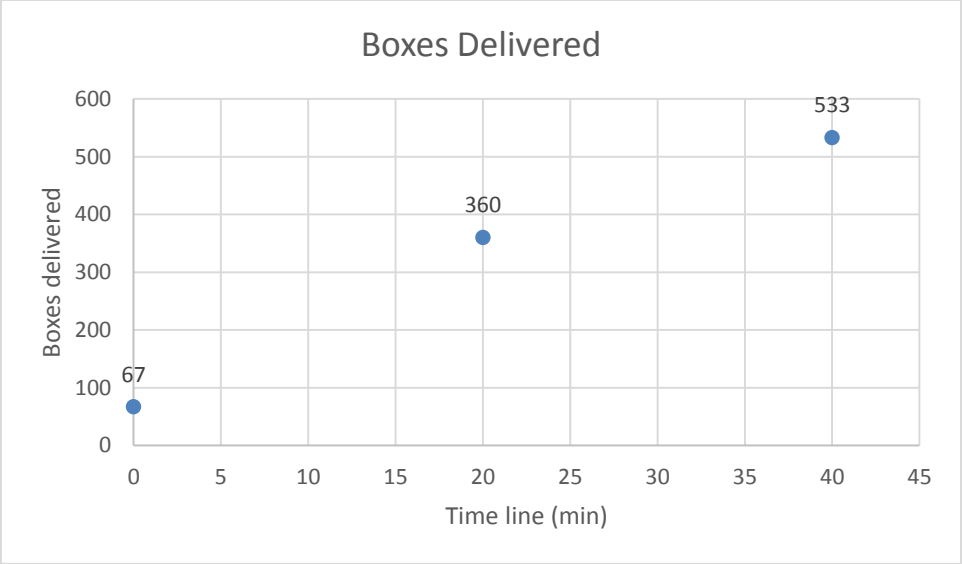

Figure 3.28: Number of boxes delivered based on the entry time of driver's

Scenario 3: Drivers entering at long gap of one-hour time intervals

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above.

Table 3.31: Input for MPMD-MD_DT simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time
D1	1	100	0
D2	8	300	60
D3	18	500	120

The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.32: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	54	87
2	36	63
3	62	52
4	27	51
5	41	50
6	76	42
7	69	31
8	69	95
9	33	60
10	26	23
11	59	26
12	78	58
13	63	11
14	51	28
15	46	26
16	34	71
17	49	65
18	41	57
19	42	19
20	50	91

In this scenario each driver enters at an interval of 60 minutes into the system and once the drivers are checked in to the system, the drivers are assigned with the routes and the following data is presented based on the number of requests, deliveries that each driver can handle based on their entry time.

Table 3.33: Total distance traveled by each driver

Drivers	Vehicle capacity	Distance	Driver Arrival Time
D1	100	1561	0
D2	300	1828	60
D3	500	1615	120

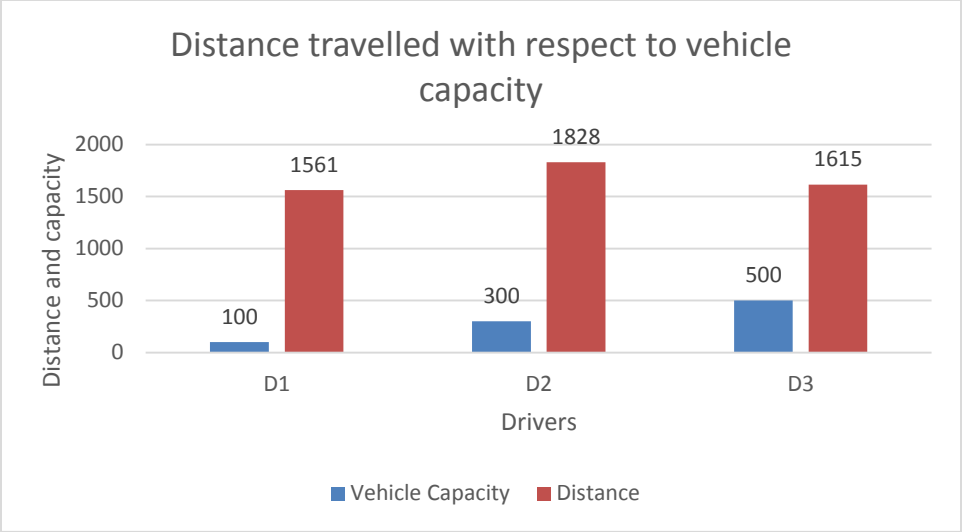

Figure 3.29: Total distance traveled by each driver

Table 3.34: Comparison of number of pickup requests handled with respect to vehicle's capacity

Drivers	Vehicle capacity	Picked requests	Driver Arrival Time
D1	100	2	0
D2	300	8	60
D3	500	10	120

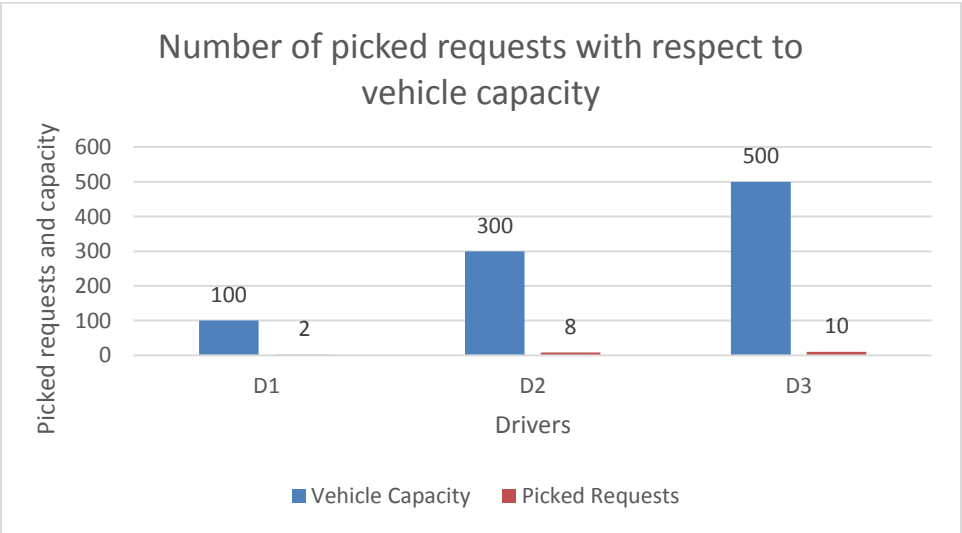

Figure 3.30: Number of pickup requests handled by each driver

Table 3.35: Comparison of number of boxes delivered with respect to the vehicle's capacity

Drivers	Vehicle capacity	Boxes delivered	Driver Arrival Time
D1	100	80	0
D2	300	336	60
D3	500	590	120

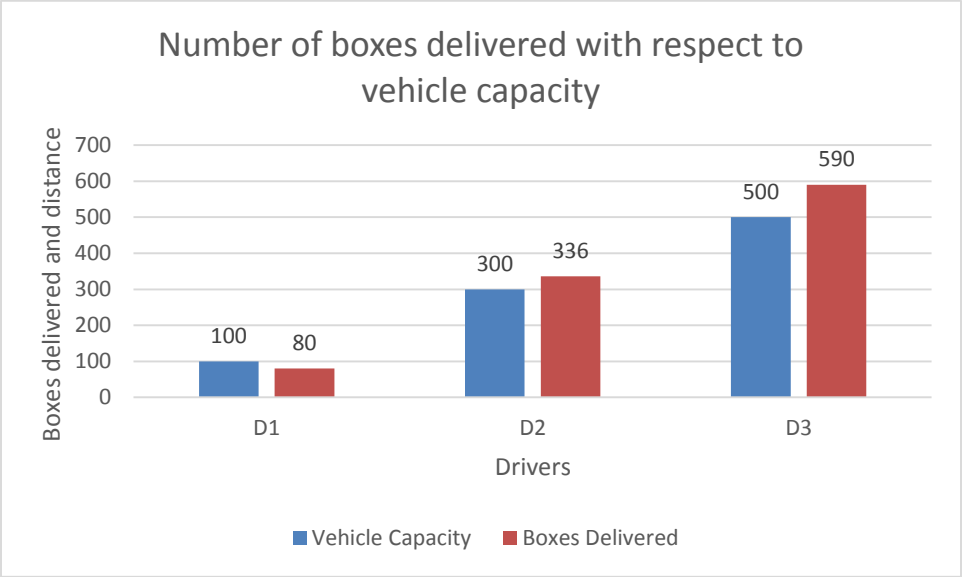

Figure 3.31: Number of boxes delivered by each driver

Table 3.36: Comparison of number of boxes delivered with respect to total distance traveled and vehicle's capacity

Drivers	Vehicle capacity	Boxes delivered	Distance	Driver Arrival Time
D1	100	80	1561	0
D2	300	336	1828	60
D3	500	590	1615	120

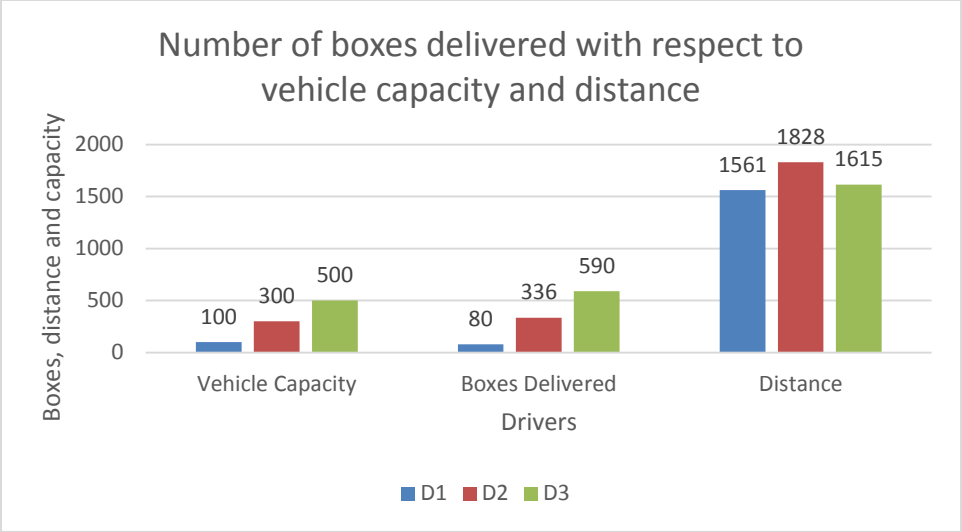

Figure 3.32: Number of boxes delivered by each driver with respect to total distance traveled and vehicle's capacity

Table 3.37: Number of boxes delivered based on driver's entry time

Drivers	time (min)	Boxes delivered
D1	0	80
D2	60	336
D3	120	590

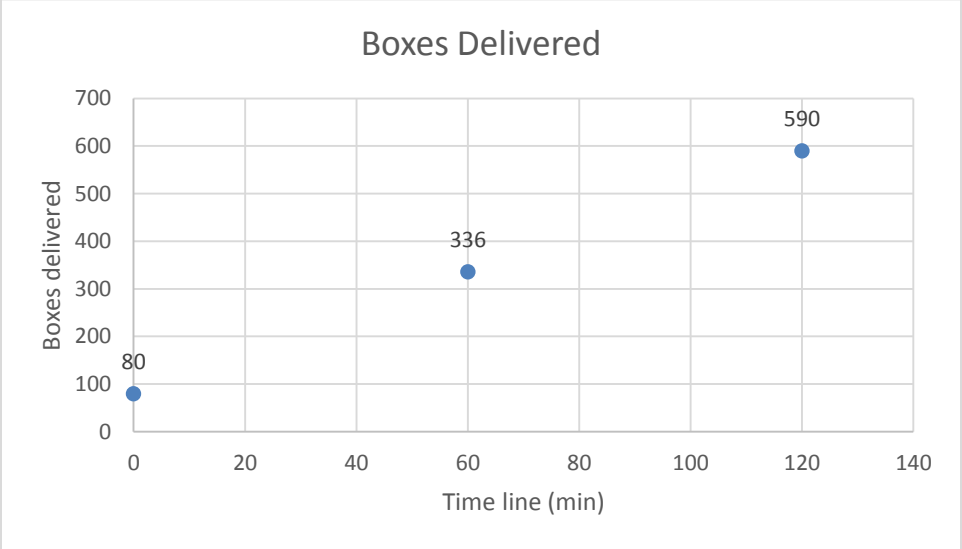

Figure 3.33: Number of boxes delivered based on the entry time of driver's

In the above figures, the driver *D1* enters first with a capacity of 100 boxes and then after some time driver *D2* enters with a capacity of 300 boxes and then later on driver *D3* enters with a capacity of 500 boxes at different time intervals. The algorithm assigns the nodes in a first-come-first-serve basis and starts assigning the nodes to driver *D1* first. Even though driver *D1* has less capacity when compared to the other drivers, he or she is able to serve more number of boxes since he or she arrived first and no other drivers were available. Later driver *D2* comes, and the remaining nodes apart from the nodes assigned to driver *D1* are distributed between drivers *D1* and *D2*. The same happens when driver *D3* arrives. Driver *D2* has delivered fewer number of boxes because he has checked in after driver *D1* and his vehicle capacity is also less compared to driver *D3*. Since driver *D3* holds more capacity to pick up boxes, he has taken more requests compared to driver *D2*.

3.4.4 Multiple pickups and multiple deliveries with multiple drivers starting at different timings and available for different shift times (MPMD – MD_DST)

In this variation, MPMD – MD_DST, i.e., Multiple Pickup and Multiple Drop-off with multiple drivers having different shift timings, a new variable, availability time is added. In MPMD – MD_DST, all the drivers are distributed sparsely around the cities and connected through an application. The driver can start his work at any time and any location. The driver should also provide the driver's availability hours along with starting location and vehicle capacity. In MPMD – MD_DST, we have considered the driver's working hours on an hourly basis. The availability time limit has not been specified. The MPMD – MD_DST will allocate nodes based on his or her availability hours. In this

variation, the drivers come at different timings and also with different availability timings. Therefore, in this version, the server needs to be running continuously to handle the upcoming drivers. The client-server approach is similar to the MPMD – MD_DT scheme in the previous section. The driver's information is taken from the client end, and the server runs through the graph (map) and available requests with the driver's information and assigns the route which maximizes the number of boxes based on his or her availability time.

3.4.4.1 Implementation of MPMD – MD_DST

MPMD_MD_DST, i.e., Multiple Pickup and Multiple Delivery with multiple drivers having different shift timings are implemented using Java's multi-threading and networking concepts. The following assumptions are made while implementing MPMD – MD_DST,

- The driver starts at one of the nodes, where there is a request to pick up boxes
- The driver is available for a certain amount of time in number of hours
- Requests are available before searching for drivers
- The distance is taken as miles and time for covering the distance is in minutes

Time is the new variable which is added to the existing modified algorithm shown in figure 3.3. The algorithm checks the time it takes to travel from one node to another, as the route assigned to the driver has to be within the driver's availability time. The driver is allowed to pick the boxes only if he can deliver those boxes within the availability time frame. Hence, at any node, there may be few pickup requests which are not serviced by a driver, but can be handled by one of the other drivers who are close to that location and

have sufficient time on their hands. Hence, unlike previous versions, it is possible that handling pickup requests more than once at the same node for different drivers will occur, based on the driver's availability. A new time matrix is taken from google maps for 20 cities. The distance matrix generated by considering the same 20 cities in Oklahoma state is shown in table 3.1, is used for time matrix below, that shows the time to travel between the same cities. This results in a new time matrix as shown below in figure 3.38.

Table 3.38: Time matrix for a graph of 20 nodes in minutes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	66	69	110	71	145	139	129	175	61	170	202	247	185	109	125	336	192	69	36
2	66	0	92	142	99	63	83	112	117	33	112	175	219	157	82	153	315	174	112	36
3	69	92	0	136	106	177	163	96	166	110	191	251	295	230	163	163	267	125	82	67
4	110	142	136	0	49	226	184	228	255	135	250	196	242	172	137	60	396	227	88	141
5	71	99	106	49	0	181	176	190	211	91	205	149	195	130	92	62	368	230	103	97
6	145	63	177	226	181	0	104	196	201	110	55	120	165	162	105	238	403	261	194	117
7	139	83	163	184	176	104	0	135	116	106	60	212	256	230	152	232	324	212	189	112
8	129	112	96	228	190	196	135	0	82	139	187	281	324	262	188	248	222	80	173	110
9	175	117	166	255	211	201	116	82	0	144	166	286	328	266	192	265	219	123	223	146
10	61	33	110	135	91	110	106	139	144	0	135	166	211	147	71	147	343	203	127	52
11	170	112	191	250	205	55	60	187	166	135	0	170	215	214	152	260	360	262	218	141
12	202	175	251	196	149	120	212	281	286	166	170	0	50	90	111	201	486	347	254	203
13	247	219	295	242	195	165	256	324	328	211	215	50	0	131	158	248	531	392	297	248
14	185	157	230	172	130	162	230	262	266	147	214	90	131	0	88	154	465	326	229	180
15	109	82	163	137	92	105	152	188	192	71	152	111	158	88	0	150	392	252	168	106
16	125	153	163	60	62	238	232	248	265	147	260	201	248	154	150	0	427	283	149	153
17	336	315	267	396	368	403	324	222	219	343	360	486	531	465	392	427	0	170	330	312
18	192	174	125	227	230	261	212	80	123	203	262	347	392	326	252	283	170	0	156	172
19	69	112	82	88	103	194	189	173	223	127	218	254	297	229	168	149	330	156	0	86
20	36	36	67	141	97	117	112	110	146	52	141	203	248	180	106	153	312	172	86	0

The system now takes both the distance matrix and its related time matrix as input along with the driver's information. The requests are randomly generated at each node, with a random number of delivery points and boxes at those delivery points. The driver's information is taken as the input from the client side, and the algorithm which runs continuously at the server side handles the information from the client side and generates the route based on these parameters.

Algorithm

The implementation of this algorithm has two parts, the client environment, and the server environment. The client environment is similar to MPMD – MD_DT, where the driver's information of starting location and vehicle capacity is taken along with driver availability hours. A new variable *driver.availability_time* is added to the driver's information in the environment as shown below in figure 3.34.

```
create the socket object for connecting to a port
    while terminate != true do
        driver.starting_location ← starting_location
        driver.vehicle_capacity ← capacity
        driver.availability_time ← availability_time
        visited[i] ← starting_location
        route[i] ← starting_location
        //sends the driver information to server side
        output_stream ← driver_object
        if drivers_available == true then
            continue
        else if drivers_available == false then
            terminate ← true
```

Figure 3.34: Client environment for MPMD - MD_DST

The server environment is similar to MPMD – MD_DT, except for a condition to check for the threshold limit on the number of boxes handled per hour. If the number of boxes handled per hour is less than 20 boxes, then the driver is sent to the waiting list and all the assigned requests are set back to the initial state. If the boxes are above 20 in number, then the driver is assigned with the route to handle the requests. The server environment of MPMD – MD_DST is shown below in figure 3.35.

```
Create the socket connection for clients to get connected
    Thread t
        while true do
            Modified ant algorithm as described in figure 4
        if driver.number_of_boxes_assigned < 20 then
            driver.reset()
    Thread main
        while trip != true do
            if driver_object_client != null then
                driver_object_server ← driver_object_client
                drivers_list ← driver_object_server
                t. start()
            else if driver_object_client == null then
                trip ← true
```

Figure 3.35: Server environment for MPMD - MD_DST

The *vehicle_loading* function is reloaded with few functionalities as mentioned below compared to the previous versions in the above sections. Since in MPMD – MD_DST we have introduced the new variable time, the algorithm needs to check the total time it takes to handle the deliveries that have been picked up by the driver. So, every time the node is selected, the travel time is also calculated, and if it comes below the total availability time, the node is added to the route, else the next best node is selected. The vehicle_loading function is shown in figure 3.36.

The *vehicle_unloading* function is similar to the previous variations as shown in figure 3.7. The following parameters are used in the implementation of the above algorithm for MPMD – MD_DST, $q \leftarrow$ random variable between 0 and 1, $q_0 \leftarrow 0.9$, $\alpha \leftarrow 0.01$, $\beta \leftarrow 4$. The values were selected based on [13][15].

```
def vehicle_loading (current_node, driver)
    if pickup_track == 1 && driver.assigned_time < driver.availability_time then
       get_deliveries ← deliveries(current_node)
       node ← current_node
    for each i ∈ get_deliveries do
       time ← time[node][i]
       val ← val + get_deliveries.get(i)
       node ← i
    if (driver.vehicle_storage + val) > driver.vehicle_capacity || (driver.assigned_time +    time) >
    driver.availability_time then
         if (driver.vehicle_storage + val) > driver.vehicle_capacity then
              vehicle_overloaded ← 1
         else continue
    else (driver.vehicle_storage + val) < driver.vehicle_capacity then
          driver.vehicle_storage ← driver.vehicle_storage + val
          pickup_track ← 0
```

Figure 3.36: Vehicle_loading function in MPMD - MD_DST

3.4.4.2 Results for MPMD – MD_DST

The results generated below are for three scenarios where all the drivers enter at different time intervals i.e., short time, mixed time and large time intervals as shown in the below tables and graphs. Also in scenario four the results are shown for multiple drivers available for different times.

Scenario 1: Drivers entering in short time gap of 3 minute intervals

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above. The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.39: Input for MPMD-MD_DST simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time	Driver Available time
D1	1	100	0	180
D2	8	300	3	300
D3	18	500	6	480

Table 3.40: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	18	34
2	105	41
3	87	66
4	104	89
5	89	71
6	44	86
7	35	88
8	68	86
9	63	63
10	108	111
11	73	47
12	32	114
13	49	62
14	44	55
15	107	128
16	72	51
17	48	72
18	116	82
19	49	68
20	130	27

In this scenario each driver enters at an interval of 3 minutes and once the drivers are checked in to the system, the drivers are assigned with the routes. The simulation results are shown below

Table 3.41: Comparison of distance traveled with respect to available time

Drivers	Available time	Distance	Driver Arrival Time
D1	180	249	0
D2	300	373	3
D3	480	489	6

Figure 3.37: Distance traveled with respect to available time by multiple drivers

Table 3.42: Comparison of handling number of picked requests with respect to available time

Drivers	Available Time	Picked Requests	Driver Arrival Time
D1	180	2	0
D2	300	1	3
D3	480	1	6

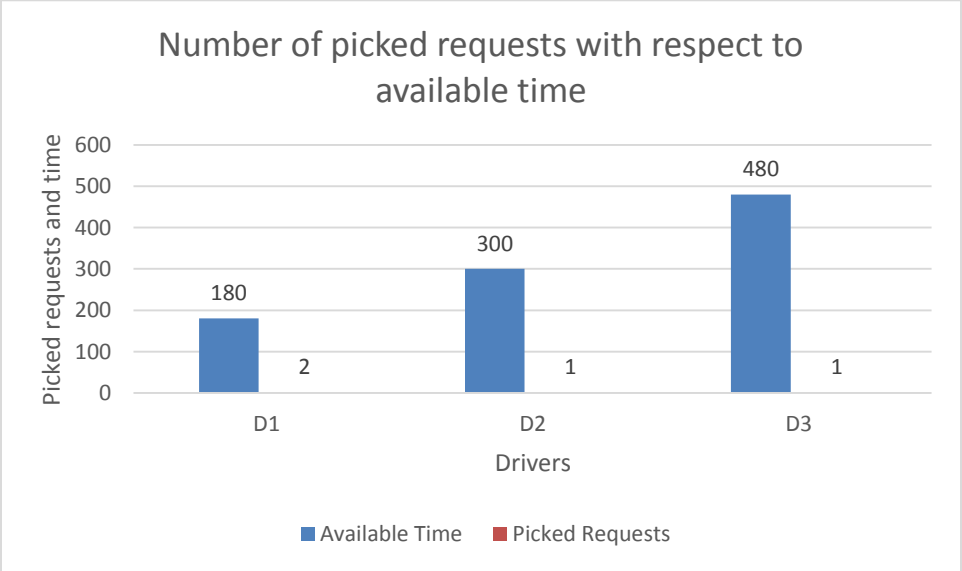

Figure 3.38: Number of picked requests by each driver

Table 3.43: Comparison of number of boxes delivered with respect to availability time

Drivers	Available Time	Boxes Delivered	Driver Arrival Time
D1	180	27	0
D2	300	40	3
D3	480	38	6

In this scenario, the number of boxes delivered by driver *D2* who is available for 3 hours of time is more than the number of boxes delivered by driver *D3* who is available for 8 hours. The total distance travelled by driver *D3* is larger compared to driver *D2* which eventually increases the total time required to handle the requests. Since driver *D3* is available for 8 hours he handled the requests through that route and the total number of boxes available at each node is generated randomly and independent of the distance travelled and time to cover the distance.

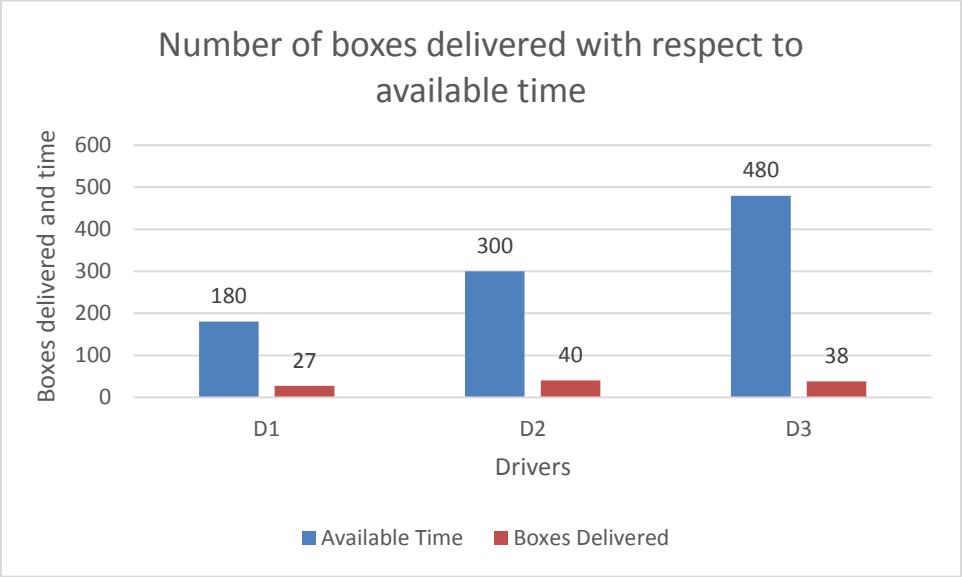

Figure 3.39: Number of boxes delivered by each driver

Table 3.44: Comparison of number of boxes delivered with respect to available time and distance

Drivers	Available Time	Boxes Delivered	Distance	Driver Arrival Time
D1	180	27	249	0
D2	300	40	373	3
D3	480	38	489	6

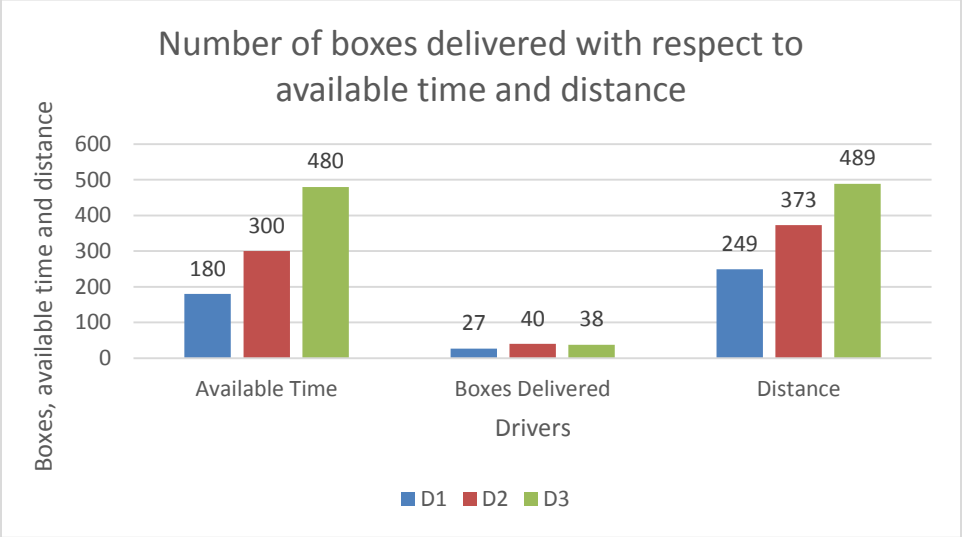

Figure 3.40: Number of boxes delivered by each driver with respect to available time and distance

Table 3.45: Number of boxes delivered based on the driver's entry time

Drivers	Time (min)	Boxes Delivered
D1	0	27
D2	3	40
D3	6	38

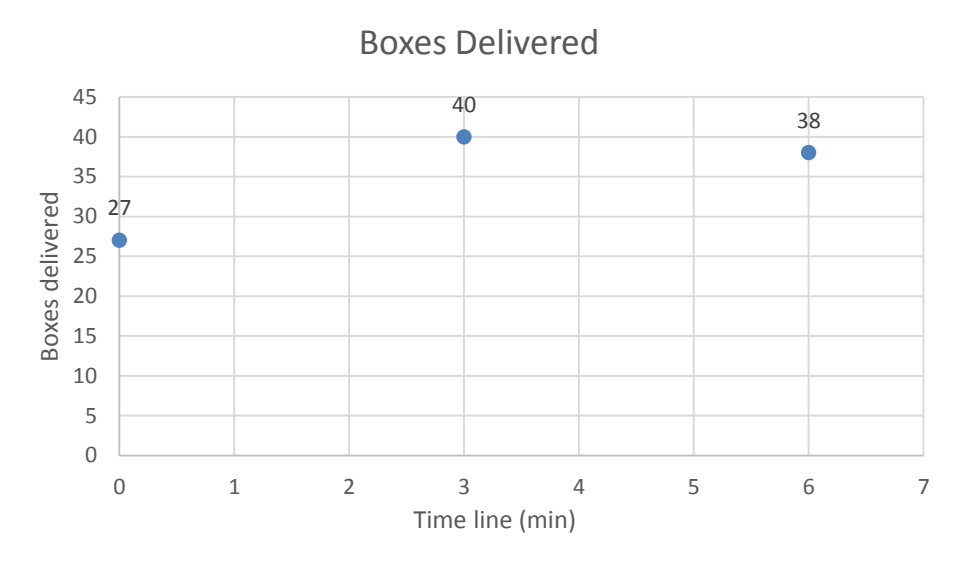

Figure 3.41: Number of boxes delivered based on the entry time of driver's

Scenario 2: Drivers entering in mixed time intervals of 20 and 40 minutes

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above. The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.46: Input for MPMD-MD_DST simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time	Driver Available time
D1	1	100	0	180
D2	8	300	20	300
D3	18	500	40	480

Table 3.47: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	125	105
2	153	107
3	121	0
4	135	92
5	72	88
6	29	111
7	81	57
8	182	78
9	38	179
10	198	146
11	191	131
12	69	115
13	37	118
14	101	110
15	75	152
16	57	150
17	131	82
18	85	78
19	60	97
20	159	103

In this scenario each driver enters in mixed interval of 20 and 40 minutes into the system and the drivers are assigned the routes. The simulation results are shown below.

Table 3.48: Comparison of total distance traveled with respect to available time

Drivers	Availability Time	Distance	Driver Arrival Time
D1	180	306	0
D2	300	250	20
D3	480	310	40

Figure 3.42: Total distance traveled by each driver

Table 3.49: Comparison of number of pickup requests handled with respect to available time

Drivers	Availability Time	Picked Requests	Driver Arrival Time
D1	180	1	0
D2	300	1	20
D3	480	2	40

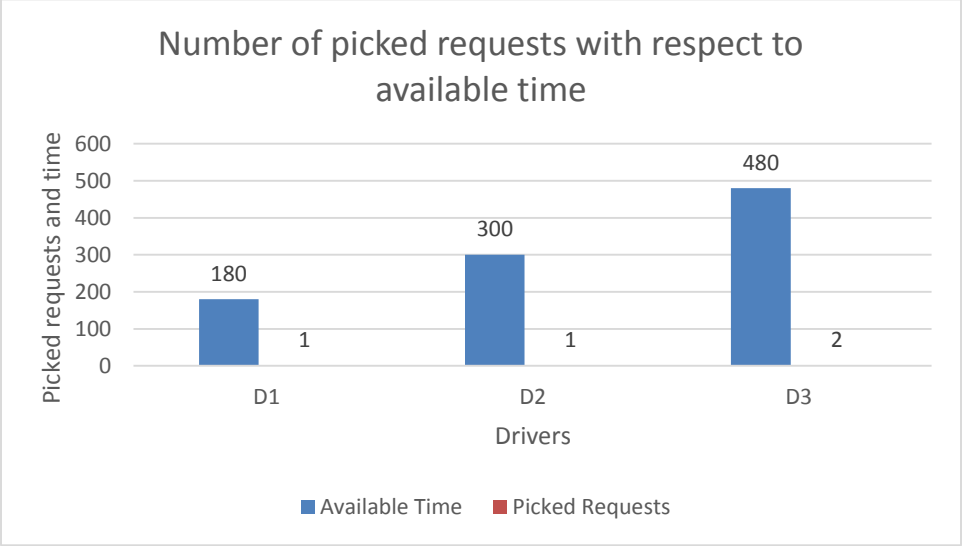

Figure 3.43: Number of pickup requests handled by each driver

Table 3.50: Comparison of number of boxes delivered with respect to available time

Drivers	Availability Time	Boxes Delivered	Driver Arrival Time
D1	180	33	0
D2	300	38	20
D3	480	36	40

Similarly, in this simulation the number of boxes delivered by driver *D2* who is available for 3 hours of time is more than the number of boxes delivered by driver *D3* who is available for 8 hours. The total distance travelled by driver *D3* is larger compared to driver *D2* which eventually increases the total time required to handle the requests. Since driver *D3* is available for 8 hours he handled the requests through that route and the total number of boxes available at each node is generated randomly and independent of the distance travelled and time to cover the distance.

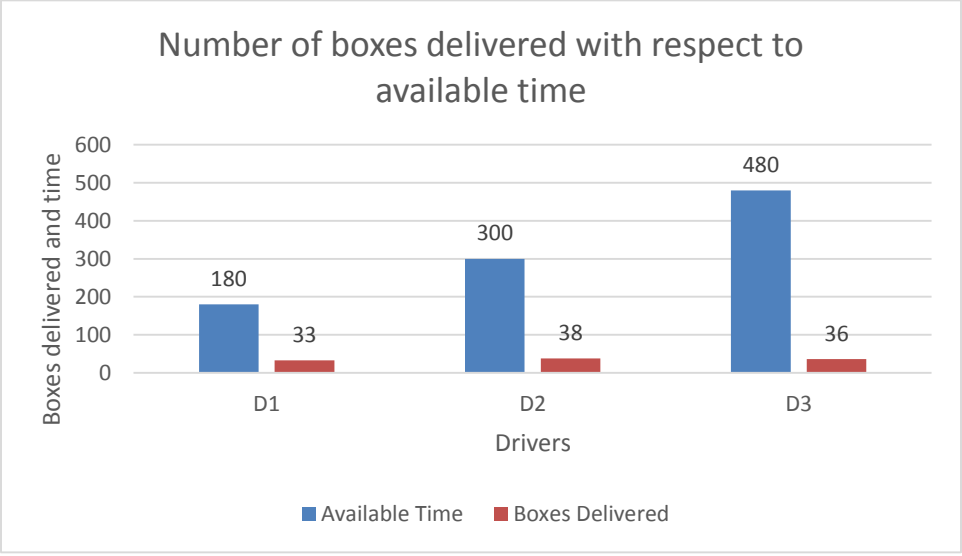

Figure 3.44: Number of boxes delivered by each driver

Table 3.51: Comparison of number of boxes delivered with respect to total distance traveled and available time

Drivers	Availability Time	Boxes Delivered	Distance	Driver Arrival Time
D1	180	33	306	0
D2	300	38	250	20
D3	480	36	310	40

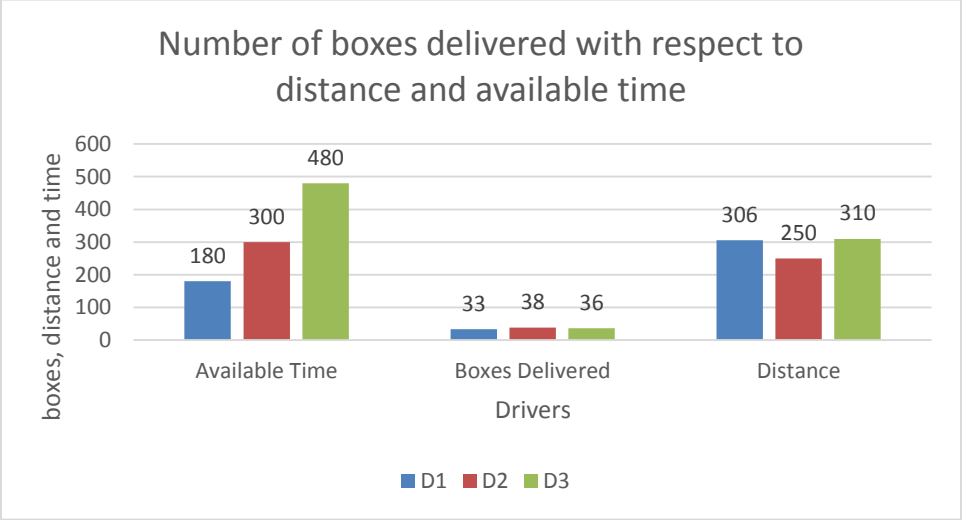

Figure 3.45: Number of boxes delivered by each driver with respect to total distance traveled and available time

Table 3.52: Number of boxes delivered based on the driver's entry time

Drivers	Time (min)	Boxes Delivered
D1	0	33
D2	20	38
D3	40	36

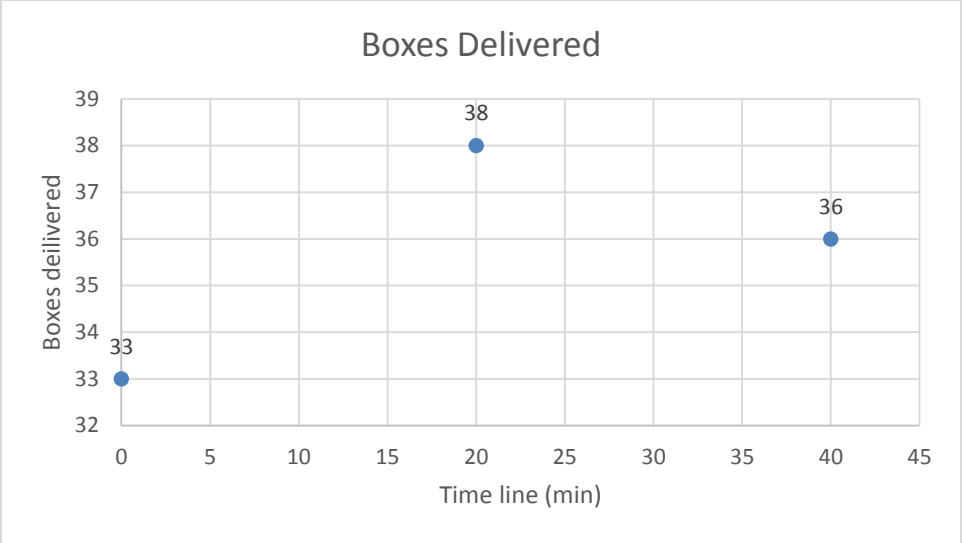

Figure 3.46: Number of boxes delivered based on the entry time of driver's

Scenario 3: Drivers entering in large time gap of one-hour time interval

The below table shows the simulation input for the below results obtained by the modified ant colony algorithm. The requests at each node are generated as mentioned above.

Table 3.53: Input for MPMD-MD_DST simulation

Driver	Starting City	Vehicle Capacity	Driver Arrival Time	Driver Available time
D1	1	100	0	180
D2	8	300	60	300
D3	18	500	120	480

The below table shows the number of boxes that need to be picked up at each node and number of boxes that need to be delivered at the same node.

Table 3.54: Requests at each node

Location	Number of boxes to be picked up	Number of boxes to be delivered
1	164	81
2	124	153
3	146	119
4	143	115
5	24	62
6	40	87
7	97	120
8	121	46
9	57	68
10	28	154
11	108	130
12	198	84
13	51	54
14	152	117
15	73	96
16	108	133
17	171	96
18	138	114
19	40	140
20	48	62

In this scenario each driver enters at an interval of 60 minutes into the system and the drivers are assigned the routes. The simulation results are shown below.

Table 3.55: Comparison of total distance traveled with respect to available time

Drivers	Availability Time	Distance	Driver Arrival Time
D1	180	288	0
D2	300	286	60
D3	480	378	120

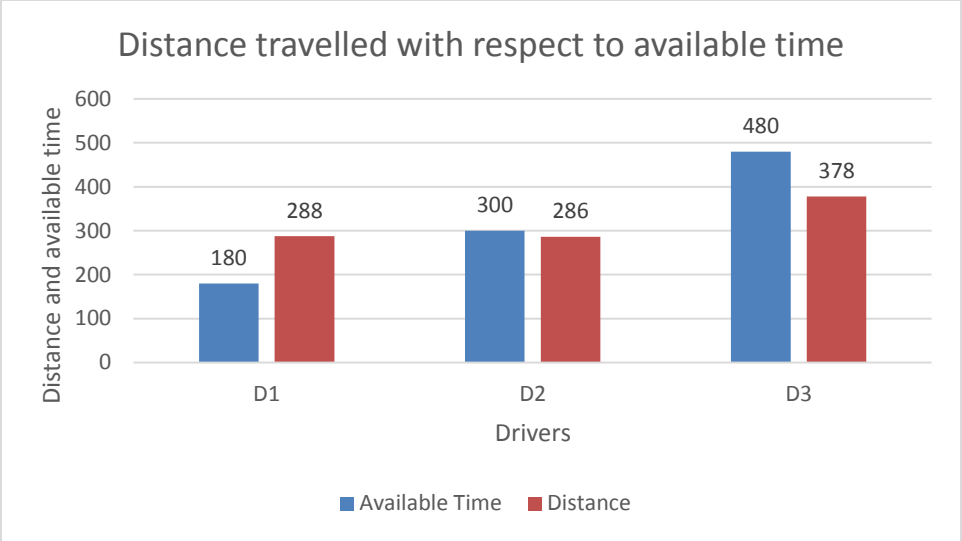

Figure 3.47: Total distance traveled by each driver

Table 3.56: Comparison of number of pickup requests handled with respect to available time

Drivers	Availability Time	Picked Requests	Driver Arrival Time
D1	180	2	0
D2	300	1	60
D3	480	1	120

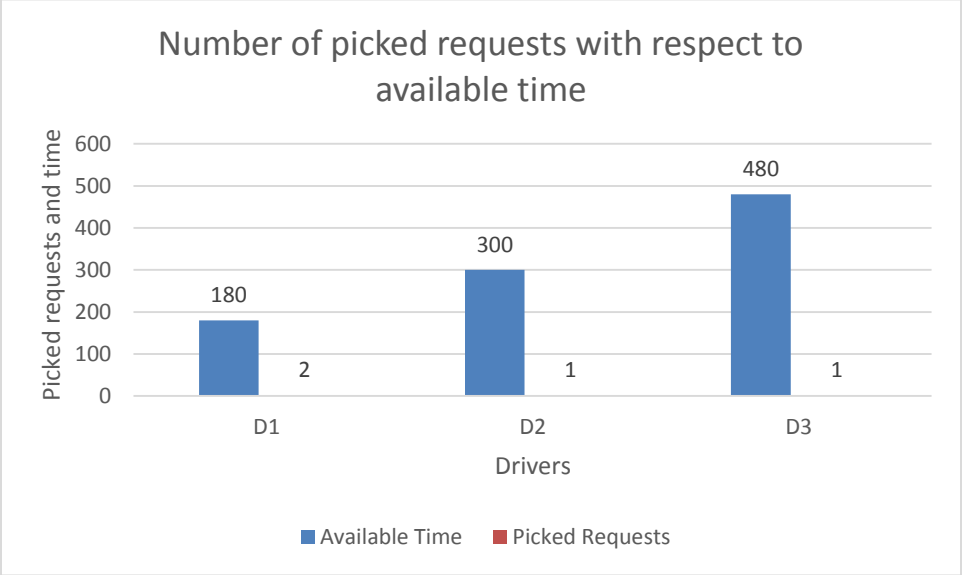

Figure 3.48: Number of pickup requests handled by each driver

Table 3.57: Comparison of number of boxes delivered with respect to available time

Drivers	Availability Time	Boxes Delivered	Driver Arrival Time
D1	180	33	0
D2	300	58	60
D3	480	41	120

Similarly, in this simulation the number of boxes delivered by driver D2 who is available for 3 hours of time is more than the number of boxes delivered by driver D3 who is available for 8 hours. The total distance travelled by driver D3 is larger compared to driver D2 which eventually increases the total time required to handle the requests. Since driver D3 is available for 8 hours he handled the requests through that route and the total number of boxes available at each node is generated randomly and independent of the distance travelled and time to cover the distance.

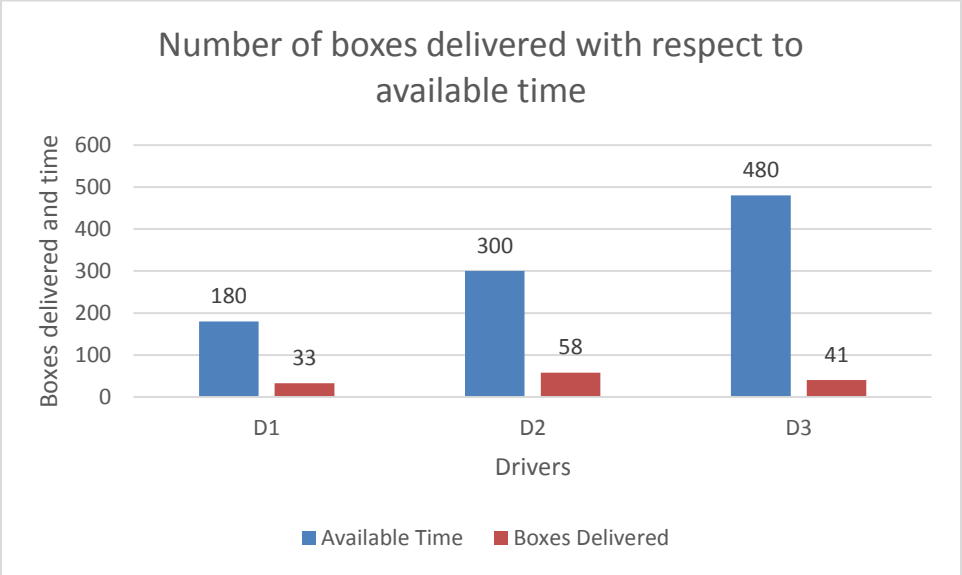

Figure 3.49: Number of boxes delivered by each driver

Table 3.58: Comparison of number of boxes delivered with respect to total distance traveled and available time

Drivers	Availability Time	Boxes Delivered	Distance	Driver Arrival Time
D1	180	33	288	0
D2	300	58	286	60
D3	480	41	378	120

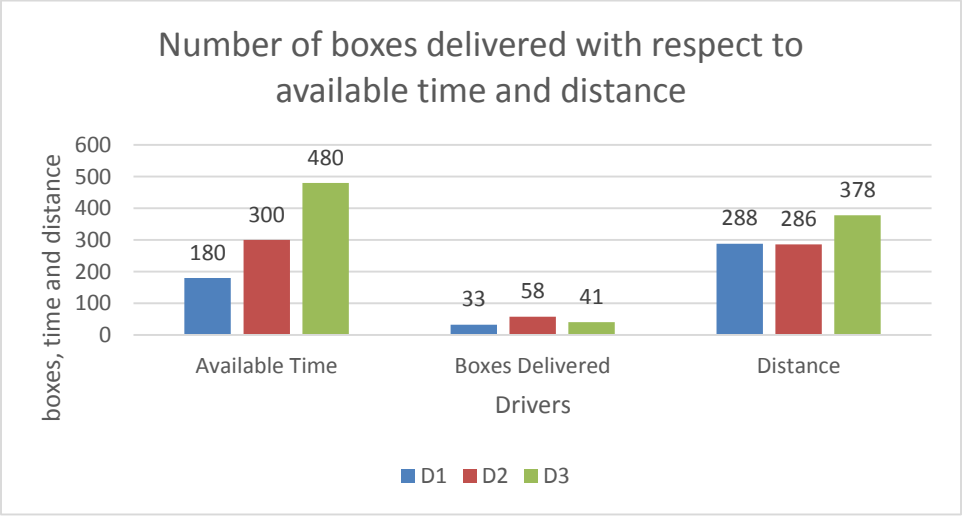

Figure 3.50: Number of boxes delivered by each driver with respect to total distance traveled and available time

Table 3.59: Number of boxes delivered based on the driver's entry time

Drivers	Time (min)	Boxes delivered
D1	0	33
D2	60	58
D3	120	41

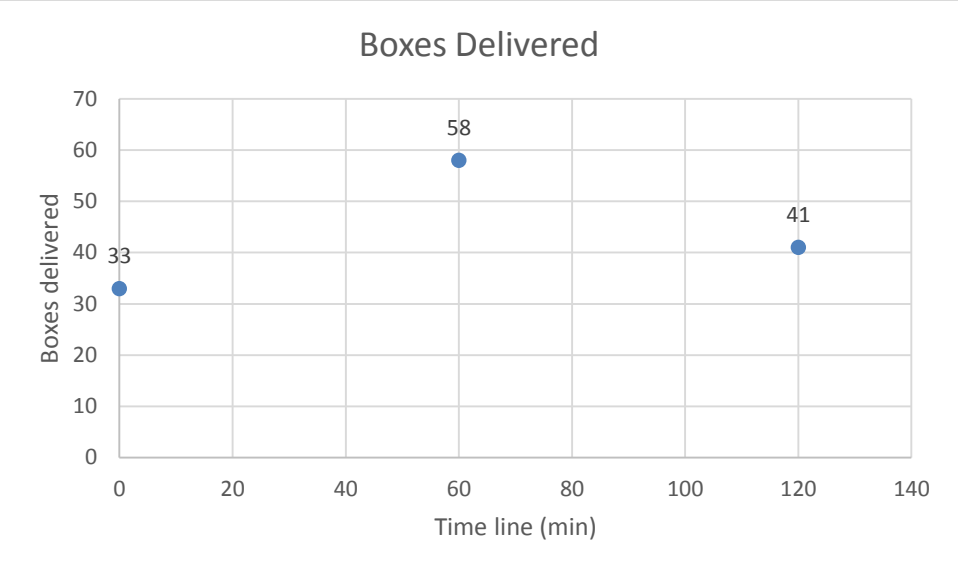

Figure 3.51: Number of boxes delivered based on the entry time of driver's

In the above figures, the requests are generated randomly as mentioned in the previous sections. The drivers start at different locations, at different timings and are available for different times. Driver *D1* is started first, later on driver *D2*, *D3* and each driver started in the different time intervals. In this simulation, each driver has different vehicle capacity.

In the above results, the requests are generated randomly as mentioned above. The number of boxes that need to be picked up and delivered at particular node is not constant, they vary for every simulation and hence the number of boxes delivered by each driver are similar and independent of time. Since Driver *D1* and *D2* has arrived before *D3* the requests are first assigned and when driver *D3* arrives the remaining requests are assigned based on availability. Driver *D2* might not cover larger distance because of his availability, but driver *D3* is able to cover larger distances and fulfil the requests at each node.

CHAPTER 4

CONCLUSIONS

The Multiple pickups and Multiple drop-off problems, with single and multiple drivers, drivers starting at same and different times, varied vehicle capacities, drivers starting at different times with different shift timings, is solved with a modified ant colony algorithm. The variations are tested for different samples of data as shown above i.e., multiple drivers, and different maps. Our approach is able to produce similar results to the brute-force approach and better results than the Nearest Neighbor algorithm. The Ant colony algorithm works efficiently by providing good results in polynomial time and also for dynamically increasing data. As far as we are aware, the work done in this thesis has not been addressed by other researchers. The results are compared with the shortest distance that each driver travels and also with the amount of time for the algorithm to run. The results are shown in terms of the number of boxes delivered and also in terms of total distance traveled by each driver with respect to vehicle capacity, and his or her availability time in the previous section. The graph in figure 4.1 show the results of our modified ant colony algorithm compared with brute-force approach and nearest neighbor algorithms. The x-axis represents the number of nodes used in the simulation and y-axis shows the shortest distance generated by the algorithms. As stated, the modified ant colony algorithm is able to generate similar results

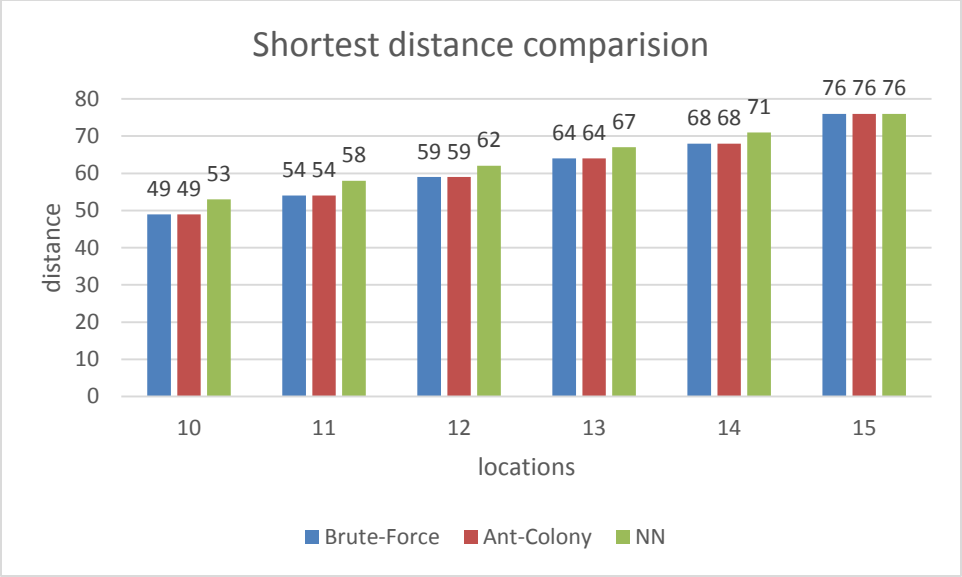

Figure 4.1: Comparison of shortest distance generated by modified ant colony algorithm with different algorithms

to brute force approach and better results than nearest neighbor algorithm.

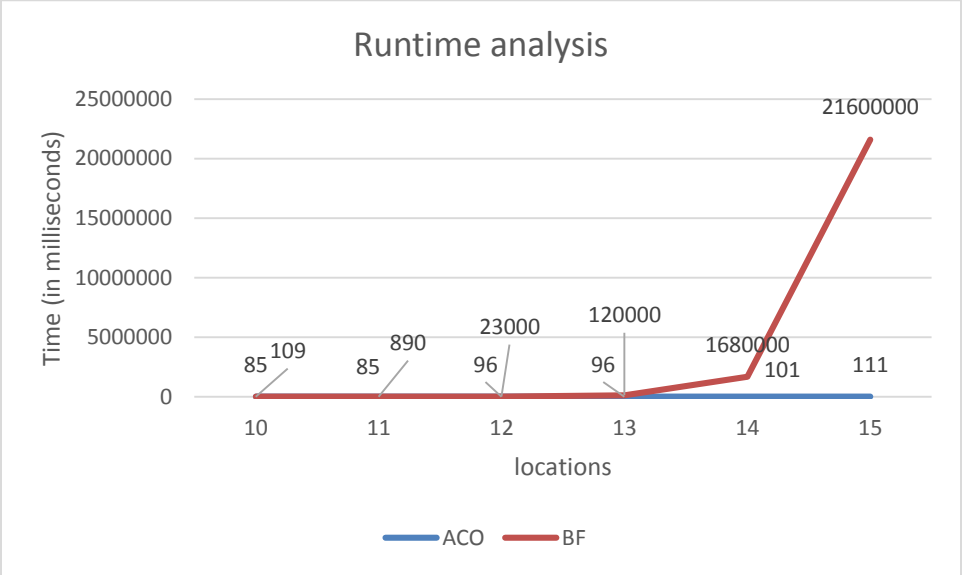

Figure 4.2: Runtime Analysis of Brute force and Ant colony

The above figure 4.2 shows the runtime analysis of modified ant colony algorithm with brute – force method and results shows that the modified ant colony algorithm is able to produce better results in polynomial time.

Future work can include further development as a mobile application with a map interface along with these functionalities. The application can be used in real-world scenarios that involve multiple pickups and multiple delivereies. Future work may also implement features like handling multiple drivers available at the same locations, i.e., near malls, and regular pickup locations in a first-in-first-out manner.

REFERENCES

[1] T. Bektas, "The multiple traveling salesman problem: An overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[2] R. Ponraj and G. Amalanathan, "Optimizing multiple traveling salesman problem considering the road capacity," *J. Comput. Sci.*, vol. 10, no. 4, pp. 680–688, 2014.

[3] S. H. Chen, "Minimization of the Total Traveling Distance and Maximum Distance by Using a Transformed-Based Encoding EDA to Solve the Multiple Traveling Salesmen Problem," *Math. Probl. Eng.*, vol. 2015, 2015.

[4] R. Krishnamurti, "The Multiple Traveling Salesman Problem with Time Windows : Bounds for the Minimum Number of Vehicles," *Time*, pp. 1–16, 2002.

[5] M. S. Hou and D. B. Liu, "A novel method for solving the multiple traveling salesmen problem with multiple depots," *Chinese Sci. Bull.*, vol. 57, no. 15, pp. 1886–1892, 2012.

[6] N. Wassan and G. Nagy, "Vehicle Routing Problem with Deliveries and Pickups: Modelling Issues and Meta-heuristics Solution Approaches," *Int. J. Transp.*, vol. 2, no. 1, pp. 95–110, 2014.

[7] B. Cheang, X. Gao, A. Lim, H. Qin, and W. Zhu, "Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints," *Eur. J. Oper. Res.*, vol. 223, no. 1, pp. 60–75, 2012.

[8] G. Erdogan, J. F. Cordeau, and G. Laporte, "The pickup and delivery traveling salesman problem with first-in-first-out loading," *Comput. Oper. Res.*, vol. 36, no. 6, pp. 1800–1808, 2009.

[9] Y. Wang, X. Ma, Y. Lao, Y. Wang, and H. Mao, "Vehicle Routing Problem Simultaneous Deliveries and Pickups with Split Loads and Time Windows," *Transp. Res. Rec.*, vol. 1500, no. 2378, pp. 120–128, 2013.

[10] M. T. Goodrich and P. Pszona, “Two-Phase Bicriterion Search for Finding Fast and Efficient Electric Vehicle Routes,” pp. 193–202, 2014.

[11] G. Gutin, A. Yeo, and A. Zverovich, “Traveling salesman should not be greedy : domination analysis of greedy-type heuristics for the TSP,” vol. 117, pp. 81–86, 2002.

[12] A. S. Rostami, F. Mohanna, H. Keshavarz, and A. A. R. Hosseinabadi, “Solving multiple traveling salesman problem using the gravitational emulation local search algorithm,” *Appl. Math. Inf. Sci.*, vol. 9, no. 2, pp. 699–709, 2015.

[13] J. E. Bell and P. R. McMullen, “Ant colony optimization techniques for the vehicle routing problem,” *Adv. Eng. Informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[14] Varshika Dwivedi, Taruna Chauhan, Sanu Saxena, Princie Agarwal, “Traveling Salesman Problem Using Genetic Algorithm,” Internation Journal of Computer Applications, 2012.

[15] Alaya Ines, Solnon Christine, Khaled Ghedira, “Ant Colony Optimization for Mixed-Objective Optimization Problems,” Evolutionary computation, Vol. 18, pp. 503-518, 2010.

VITA

Gorthi, Venkata Sreeram Phani Sai

Candidate for the Degree of

Master of Science

Thesis: ANT COLONY APPROACH FOR MULTIPLE PICKUP AND MULTIPLE DROPOFF

Major Field: COMPUTER SCIENCE

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2017.

Completed the requirements for the Bachelor of Technology in Computer Science at GITAM University, Visakhapatnam, Andhra Pradesh, India in 2013.