IRis

Northeastern University

Operations Research Master's Theses

Department of Mechanical and Industrial Engineering

January 01, 2008

Time dependent truck routing and driver scheduling problem with hours of service regulations

Vidit Divyang Shah
Northeastern University

Recommended Citation

Shah, Vidit Divyang, "Time dependent truck routing and driver scheduling problem with hours of service regulations" (2008). *Operations Research Master's Theses.* Paper 2. http://hdl.handle.net/2047/d10016995

This work is available open access, hosted by Northeastern University.

Time Dependent Truck Routing and Driver Scheduling Problem with Hours of Service Regulations

A Thesis Presented

by

Vidit Divyang Shah

to

The Department of Mechanical and Industrial Engineering

In partial fulfillment of the requirements

For the degree of

Master of Science

In the field of

Operations Research

Northeastern University

Boston, Massachusetts

December 19, 2008

Abstract

Driver fatigue is the cause of many truck crashes taking place today. In order to provide safety to the truck driver, the Federal Motor Carrier Safety Administration (FMCSA) has implemented revised Hours of Service (HOS) regulations effective on October 1st 2005. According to the revised rules, drivers should take a break of 10 consecutive hours after 11 hours of driving time or 14 hours of duty time that includes driving time and service time, such as loading and unloading cargo. This has caused a severe increase in the cost for the trucking industry which in turn hurt shippers and ultimately customers. Due to periodic shortage of truck drivers and the new HOS regulations, the problem of scheduling drivers has been aggravated. In addition to that, delays in picking/unloading at the service center and stiffer fines/penalties resulting from new HOS regulations has decreased productivity in the trucking industry.

Traffic congestion is ignored in most of the vehicle routing literature. In this Thesis, traffic congestion is modeled by using time dependent travel speed along each segment of the transportation network, which in turn changes the travel time required to reach the destination. Thus travel speed variability of different highway segments over the course of a day is explicitly considered in the evaluation of routes. A systematic way of routing vehicles to avoid traffic congestion or waiting at customer's location as well as coordinating timely mandatory breaks will improve both safety of truck transportation and efficiency of distribution supply chains.

A single truck routing model with time dependent travel times, HOS regulations, multiple delivery locations and service time windows is formulated in this Thesis. The objective is to minimize the total time required to serve a given number of customers.

The components of the total time are travel time, break time, waiting time and service time at the customer nodes. A driver is not allowed to start service before the beginning of the service time window of the customer; he/she waits at the node or takes a daily break before the start of service. In addition, it has been assumed that once the driver starts service at a location he/she is not allowed to take a daily break before the completion of service. The path taken by the truck from a customer k to a customer l depends on the departure time from node k and is found by implementing the time dependent Dijkstra's algorithm that incorporates HOS regulations. An initial tour is constructed by a greedy heuristic.

Simulated Annealing (SA) metaheuristic has been applied to most of the combinatorial optimization problems. We chose SA as the solution procedure for our problem because of its ability to escape from local minima and its easy adaptability to the problem. SA accepts uphill task with certain probability which allows the heuristic to escape from local minima.

The SA heuristic is adapted to obtain near optimal solutions to our problem. The accuracy comparison was obtained by finding the optimal solution with explicit enumeration on small size problems of up to 10 customers. Our experimental results show that the heuristic gives good quality solutions (average deviation is less than 1% from the optimal objective value) in reasonable computation time. The methodology can be easily modified and adapted to more complex problems involving multiple capacitated vehicles with pickup and delivery.

Acknowledgments

I would like to thank my thesis advisor Professor Melachrinoudis for his support, valuable time throughout my thesis and graduate studies. His invaluable guidance, and encouragement that he provided during my Thesis kept my spirit and encouraged me to complete it.

I am thankful to all my friends who directly or indirectly helped me in completing my Thesis. I am indebted to my friend Fredie Terence for taking time out from his busy schedule, providing me insight and his guidance for coding the algorithm.

Dedication

I am grateful to many people who have taught me lessons in different stages of my life. I am thankful to my parents Divyang Shah and Swati Shah and my brother Dharit Shah for encouraging me, supporting me in my worst times, giving me love and trusting my capabilities when I needed the most. I am thankful to my love, Vacha Shah for her emotional support, accompanying me in my difficult times, entertaining me and finally my reason to smile.

My grandmother passed away while I was working on my thesis. I would like to dedicate my work to my grandmother and parents.

CONTENTS

ABSTRACT IV
ACKNOWLEDGMENTS V
DEDICATION VI

CONTENTS VII

LIST OF TABLES VIII
LIST OF FIGURES IX

CHAPTER 1 1

INTRODUCTION 1

THESIS MOTIVATION 1
THESIS ORGANIZATION 2

CHAPTER 2 4

LITERATURE REVIEW 4

2.1 TRAVELING SALESMAN PROBLEM (TSP) 4
2.2 VEHICLE ROUTING PROBLEM WITH HOURS-OF-SERVICE (HOS) REGULATIONS 5
2.3 TIME DEPENDENT VEHICLE ROUTING PROBLEM (TDVRP) 12
2.4 SIMULATED ANNEALING (SA) 18

CHAPTER 3 22

MATHEMATICAL MODEL 22

3.1 PROBLEM SPECIFICATION AND MODEL CONSTRUCTION 22
3.1.1 TIME DEPENDENT TRAVEL 23
3.1.2 MATHEMATICAL MODEL 25

CHAPTER 4 29

SOLUTION PROCEDURE 29

4.1 TIME DEPENDENT DIJKSTRA'S ALGORITHM (TDD) 31
4.2 SOLUTION IMPROVEMENT WITH SIMULATED ANNEALING 40

CHAPTER 5 46

COMPUTATIONAL RESULTS 46

5.1 SMALL PROBLEM SIZE COMPARISON 46
5.2 PERFORMANCE IN LARGER SIZE PROBLEMS 49

CHAPTER 6 55

DISCUSSION AND CONCLUSIONS 55

REFERENCES 57

APPENDIX A 60

APPENDIX B 83

APPENDIX C 114

List of Tables

5.1. Comparison of enumeration and simulated annealing results....……48

5.2. Comparison of results using different initial solutions and cooling schedules……...51

List of Figures

3.1. Speed distribution……..24

3.2. Travel time distribution…….....25

4.1. Overview of solution procedure……..29

4.2. Time dependent travel function with HOS regulations…….…….....33

4.3. Service function at a node …….39

5.1. Map of Northeast region of U.S……..……50

5.2. Travel time distribution……..53

5.3. Computation time for larger size problems…….……..53

CHAPTER 1

INTRODUCTION

Logistics is an essential part in modern life and its importance is becoming more significant everyday. The role of logistics is not only limited to commercial applications but also holds an important part in day to day living. Transportation of goods from warehouse to distribution centers as well as delivery of consumer goods such as foods, soft drinks and gasoline in large quantities daily to supermarkets and gas stations across the country is done using trucks.

Thesis Motivation

FMCSA (Federal Motor Carrier Safety Administration) implemented the new Hours of Service (HOS) regulations in order to permit drivers to have enough off-duty time to reduce fatigue caused due to driving and inconsistent sleeping patterns. The new HOS regulations may lead to substantial increase in the cost of shipping. According to (WERC, 2004), Truck industries need to hire 84,000 more drivers in order to comply with new rules as drivers cannot work unless they have accumulated enough off-duty time. Motor carriers such as Schneider National have estimated reduction in productivity by 4-19% due to implementation of the new rules. In addition, stiffer fines/penalties (between $500 and $11,000 per violation) have been imposed by the new rules due to potential loading and unloading delays in service. Wal-Mart also estimated an increase of $25 million in expense due to implementation of the new rules (Clair and Fox, 2004).

Moreover, in a congested urban environment, variation in the travel speed occurs due to fluctuation in traffic density in rush hours, bad weather, accidents, road blockage, etc. Due to above uncertainities and unforeseen events, the time to reach a destination is uncertain. As penalties have been imposed as a result of violations of the HOS regulation, any delays at customer nodes should be minimized. The solutions obtained without consideration of the above uncertainties may give sub-optimal solutions or even sometimes infeasible solutions.

We can develop feasible vehicle route by taking into consideration, the following factors:

1. Variable vehicle travel time between locations depending on traffic conditions during the day.
2. Weekly/Daily break can be taken at any places, while frequency and timing of rest stops or mid-afternoon naps can be minimized.
3. Develop the truck routes and schedules to maximize on time deliveries, sleeping/recovery time at home and minimizing non-revenue miles.
4. Minimize time for loading and unloading at pickup and delivery locations, night driving and chances for starting workweek with tired drivers.

Thesis Organization

This thesis is organized as follows: In Chapter 2 we review the early studies on the Traveling Salesman Problem (TSP), Hours of service (HOS) Regulation, Time Dependent Vehicle Routing Problem (TDVRP), the Simulated Annealing (SA) heuristic and the application of Simulated Annealing in vehicle routing problems. We introduce

the mathematical model for Time Dependent Vehicle Routing Problem with HOS regulations in Chapter 3. The solution procedure and application of the SA heuristic to our TDVRP problem is explained in detail in the following Chapter 4. For small sized problems where exhaustive enumeration is possible, the results obtained from the enumeration method and from the heuristic are compared in Chapter 5. The two methods are evaluated for both computation time and the solution quality. Computational experiments have been conducted for larger size problems using SA. The thesis ends with discussions and conclusions in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we review the Traveling Salesman Problem (TSP), the Traveling Salesman Problem with Time Windows (TSPTW), solution methodologies to solve the vehicle routing problem (VRP) with Hours-of-Service (HOS) regulations and time-dependent VRP (TDVRP). Also application of heuristic methods to solve VRP and TDVRP are presented here. We have also looked into the application of Simulated Annealing in TSP and the settings of its basic parameters are discussed in the following sections.

2.1 Traveling Salesman Problem (TSP)

Given n cities, the problem is to find the least cost path to visit all cities. Each city should be visited only once. In Logistics, one of the nodes is the depot; the tour starts and ends at the depot visiting all the customers once. The size of the solution space is $(n\text{-}1)!\,/2$, if the distance between the cities is symmetric while it is $(n\text{-}1)!$, if the distance between the cities is asymmetric, where n is the number of customers.

TSPTW is a special case of TSP where each customer node has a time window to perform service. Many approaches have been proposed to solve this problem. Baker (1983) developed an exact algorithm to solve the time constrained traveling salesman problem using a branch and bound approach. A dynamic programming strategy to solve TSP with time and precedence constraints was built by Bianca (1998). A new elimination

test to improve the results obtained by the dynamic programming approach was developed by Dumas *et al*. (1995) while Calvo (2000) used a two phase approximation algorithm to solve TSPTW. The first phase is the construction phase while the second phase is the improvement phase. In the first phase, an initial solution is obtained by assignment relaxation and a greedy insertion procedure with local search is used in the second phase to obtain an improved solution.

Heuristic approaches have been adopted to solve TSP because of their ability to cope with the complexity of the problem in a reasonable amount of time. A heuristic approach is divided into two phases: (1) The first phase deals with initial construction of a tour and (2) given an initial solution, the second phase explores the solution space by using various methods to improve the solution. Various metaheuristic techniques such as Genetic algorithm, Simulated Annealing, Tabu Search, and k-opt method have been used to solve the TSP.

2.2 Vehicle Routing Problem with Hours-of-Service (HOS) Regulations

Although a significant literature exists for the Vehicle Routing Problem (VRP), very few papers incorporate into it route and schedule realistic constraints on drivers' off-duty and driving times. Hours-of-Service (HOS) regulations were introduced in 2003 and revised in 2005 by the Federal Motor Carrier Administration to mandate legal working hours of a driver in an attempt to promote safe driving and reduce accidents. The HOS rules, effective October 1, 2005 mandate that a driver:

- May drive a maximum of 11 hours after 10 consecutive hours off duty.

- May not drive beyond the 14th hour after coming on duty, following 10 consecutive hours off duty.
- May not drive after 60/70 hours on duty in 7/8 consecutive days. A driver may restart a 7/8 consecutive day period after taking 34 or more consecutive hours off duty.
- Commercial Motor Vehicle (CMV) drivers using a sleeper berth must take 10 hours off-duty, but may split sleeper-berth time into two periods provided neither is less than 2 hours.

Since then, a very few papers have considered explicitly HOS rules in routing and driver scheduling. Xu et al. (2003) has formulated a practical pickup and delivery problem (PPDP) using multiple vehicle types that are available to fill a set of pickup and delivery orders having multiple pickup and delivery time windows. In addition to HOS rules, vehicle capacity, vehicle compatibility to orders and compatibility between orders, as well as nested precedence relationship in truck loading and unloading (LIFO) are taken into consideration. The cost of a trip is determined by several factors, such as fixed charge, total mileage, total waiting time and total layover of a driver. The notation and the set partitioning formulation of PPDP follow:

S_{kh} = is set of all feasible trips of vehicle type $h \in H$ and carrier $k \in K$

$S = \cup_{kh} S_{kh}$ is set of all feasible single-vehicle trips

V_{kh} = number of vehicles type $h \in H$ belonging to carrier $k \in K$

N = set of various pick –up and delivery locations

f_s = cost of trip $s \in S$

e_{sj} = 1, if order $j \in N$ is covered by trip $s \in S$, otherwise, 0;

x_s = 1, if trip $s \in S$ is considered in the solution; otherwise, 0;

$$Minimize \sum_{s \in S} f_s x_s \tag{2.1}$$

subject to

$$\sum_{s \in S} e_{sj} x_s = 1, \text{ for } j \in N \tag{2.2}$$

$$\sum_{s \in S_{kh}} x_s \leq V_{kh}, \text{ for } k \in K \text{ and } h \in H \tag{2.3}$$

$$x_s \in \{0,1\}, \quad \text{for } s \in S \tag{2.4}$$

The objective function (2.1) minimizes the total cost of covering the orders. Constraint (2.2) ensures that each order is covered by a trip. Constraint (2.3) ensures that there are enough vehicles of type *h* and carrier *k* to cover all the trips and constraint (2.4) specifies that decision variables are binary. The HOS rules and the other real-world complications stated above are not shown explicitly in the formulation but they are imbedded in the columns of the formulation corresponding to single-vehicle trips.

The linear relaxation of the above problem is solved by the column generation procedure. An initial number of columns are generated (corresponding to single-order trip) and the linear relaxation of the problem (Master Problem) is solved first to determine the dual variables of constraints (2.2) and (2.3). Using these dual variables, new columns corresponding to each (*k*, *h*) combination are evaluated by solving Subproblem *kh*. Heuristic procedures are used to solve the subproblems implicitly by generating new trips from existing trips. The heuristic approaches are evaluated from lower bounds obtained by solving the subproblems to optimality using an exact dynamic programming algorithm.

Portugal, Ramalhinho-Laurenco and Paixao (2006) used a Set Partition Problem/ Set Covering Problem formulation (SPP/SCP) to model the driver scheduling problem. They solved the problem in two phases: (a) Generation Phase and (b) Resolution Phase. The generation phase consists of the feasible set of duties based on the parameters

defined. Labor rules, security procedures, planning strategies can be used to generate a set of feasible duties during the generation phase. In the Resolution Phase, a subset of feasible duties is selected on a particular schedule to minimize cost.

Goel and Gruhn (2006) developed a model using driver's working hours regulation in the European Union. European Union has a more complicated set of rules as compared to the U.S Department of Transportation HOS Regulations. The new European Union drivers' working hours regulations are as follows:

1. After a driving period of four and half hours, a driver shall take an uninterrupted break of at least 45 minutes unless he/she has taken a rest period.
2. The daily driving time between rest periods should be at most 9 hours. A daily rest period is any period of at least 11 hours.
3. The weekly driving time should not exceed 56 hours.
4. The weekly rest period shall start no later than 144 hours after the end of the previous week's rest period.
5. In the multiple manned vehicles, the other driver(s) can take break on the moving vehicle. In this case, the daily rest period in which the vehicle must be stationary may be reduced to 9 hours.

In the paper, the authors have not taken into consideration the following regulations:

1. The daily driving time can be extended to at most 10 hours not more than twice during the week.
2. The daily rest period may be reduced to 9 hours not more than 3 times during the week.
3. The break of 45 minutes may be replaced by one break of 15 minutes followed by other break of 30 minutes.

4. The daily rest period may be taken in two periods, the first of which must be an uninterrupted period of at least 3 hours and the second an uninterrupted period of at least 9 hours.
5. Within each period of 24 hours after the end of the precious daily rest period a driver shall have taken a new daily rest period.

Furthermore, there are regulations regarding weekly rest periods, the maximum weekly working time, and the accumulated driving time during any two consecutive weeks.

The drivers' working rules are embedded into a vehicle routing problem with time windows and capacity constraints. Let C denote the set of customer locations and V denote the set of vehicles available. For each vehicle let $n_{(v,1)}$ and $n_{(v,2)}$ denote a node corresponding to the depot where all vehicle v starts and ends its tour. Let

$D_+ = \{n_{(v,1)} \mid v \in V\}$

$D_- = \{n_{(v,2)} \mid v \in V\}$

N: $C \cup D_+ \cup D_-$

A: $(C \cup D_+) \times (C \cup D_-) / \{(n, n) | n \in C\} \cup (D_- \times D_+)$

c_{nm} is the cost and d_{nm} is the travel time associated with arc (n, m). The capacity of each vehicle is denoted by r^{max} and each node n has a demand r_n. The time window associated with each customer n is denoted by $[t_n^{\min}, t_n^{\max}]$. For each arc $(n, m) \in A$, the binary variable x_{nm} indicates whether node m is visited immediately after node n. For each node $n \in N$ the variable ρ_n represents the accumulated demand.

The Vehicle Routing Problems with Time Windows is formulated as follows:

$$\text{Min} \sum_{(n,m)\in A} c_{nm} x_{nm} \tag{2.5}$$

Subject to

$$\sum_{(n,m)\in A} x_{nm} = \sum_{(m,n)\in A} x_{mn} \quad \text{for all n} \in \text{N} \tag{2.6}$$

$$\sum_{(n,m)\in A} x_{nm} = 1 \quad \text{For all } n \in N \setminus D_{-} \tag{2.7}$$

$$\rho_n = 0 \quad \text{for all } n \in D_{+} \tag{2.8}$$

for all $(n,m) \in A$ with $n \in C \cup D_{+}$ (2.9)

if x_{nm} =1 then $\rho_m = \rho_n + r_m$ (2.10)

$$\rho_n \le r^{\max} \quad \text{for all } n \in N \tag{2.11}$$

For all $(n, m) \in A$ with $n \in C \cup D_{+}$ (2.12)

if x_{nm}= 1 then $t_m \ge t_n + d_{nm}$ (2.13)

$$t_n^{\min} \le t_n \le t_n^{\max} \quad \text{for all } n \in N \tag{2.14}$$

$$x_{nm} \in \{0,1\} \quad \text{for all } (n,m) \in A \tag{2.15}$$

Equation (2.6) states that exactly the same numbers of vehicle arrive and depart at any node $n \in N$. Equation (2.7) imposes that each node is visited exactly once. Constraints (2.8), (2.9), (2.10) are the capacity constraints which impose that the accumulated demand at any point in the tour is less than or equal to the capacity of the vehicle. Constraints (2.13) and (2.14) are the time window constraints, which imposes that each arrival time is within the time window at the node. Constraint (2.15) imposes that all x_{nm} are binary.

While considering drivers' working hours, the total travel time of the trip is the sum of driving time and the time taken for rest/break. The following notation is used in describing the drivers' working rules:

s_n = the service time required at node $n \in N$

δ_{nm} = the pure driving time from node $n \in N$ to node $m \in N$.

t_{weekly} = The maximum weekly driving time between two consecutive weekly rest periods.

t_{daily} = The maximum daily driving time between two consecutive daily rest periods.

$t_{nonstop}$= The maximum nonstop driving time between two consecutive breaks or rest periods.

t_{rest} = the time required for a daily rest periods.

t_{break} = the time requires for a break.

The state of the driver at any given node can be represented by

$$l_n = \begin{cases} l_{n,1} & \text{Arrival Time} \\ l_{n,2} & \text{Weekly Driving Time} \\ l_{n,3} & \text{Daily Driving Time} \\ l_{n,4} & \text{Nonstop Driving Time} \end{cases}$$

Consider that the vehicle travels from node *n* to node *m*. Then the state of the driver at node m can be described in terms of the state of the driver at node *n* as follows:

$$l_m = (l_{n,1} + s_n, l_{n,2} + \delta_{nm}, l_{n,3}, l_{n,4})^T$$

Let $l_m \in L$ be a label generated. The arrival time $l_{m,1}$ may be smaller then the beginning of the time window $t_m^{\min}$. Therefore, for all $l_m \in L$ let

$$L(l_m) = \{ (\max\{ t_m^{\min}, l_{m,1}, l_{m,2}, l_{m,3}, l_{m,4})^T,$$

$$(\max \{ t_m^{\min}, l_{m,1} + \text{break}\}, l_{m,2}, l_{m,3}, 0)^T,$$

$$(\max \{ t_m^{\min}, l_{m,1} + \text{t}_{\text{rest}} \}, l_{m,2}, 0,0)^T \}$$

The above set denotes the set of possible labels. The large neighborhood search algorithm is used to obtain the optimal solution, i.e, *k* customers are removed from the tour and an

insertion method tries to re-insert *k* nodes back into the tour at every iteration. If the new solution obtained is better than the current solution, it replaces the current solution. Once the termination criterion is met, the best solution is displayed.

Archetti and Savelsbergh (2007) used HOS rules in the trip scheduling problem (TSP). Given a sequence of *n* transportation requests with dispatch windows at the origins, the TSP determines the driver's schedule (if one exists), i.e., driving times and rest times, so that the origins are visited in the given sequence and within their dispatch windows, or otherwise it is found that such a feasible schedule does not exist. They assume that loading and unloading at pickup and delivery locations is done instantaneously, waiting time at the locations can be converted into daily break time; and travel time between locations is fixed instead of variable time depending on traffic and the time of the day. They develop a O (n^3) polynomial time backward search algorithm to solve the problem. The authors emphasize that HOS rules severely restrict the feasible driver schedules.

2.3 Time Dependent Vehicle Routing Problem (TDVRP)

Malandraki and Daskin (1992) proposed a mixed integer linear programming formulation of TDVRP (Time Dependent Vehicle Routing Problem) that considers travel time as step function, time windows for serving the customer and the maximum allowable duration for each route (i.e. workday of the driver). The distance of arc (i, j) is a time dependent function as the speed of the vehicle does not remain constant due to the variable traffic density. Following are the assumptions for the problem:

1) The travel time in traversing arc (i, j) during a particular time interval, is independent of the type of the vehicle used.
2) The cost of the tour is not probabilistic.
3) The service time at nodes is independent of the size of the vehicle.

Problem Formulation:

Notation used in the formulation is summarized below:

n = number of nodes including depot.

M= number of time intervals considered for each link

K = number of vehicles

c_{ij}^{m} = travel time from node i to j if starting at i during time interval m; $c_{11}^{m} = \infty$ for all i, m

c_i = service time at node i (e.g. unloading time); c_i =0 for i = 1, n+1, ……, n+K where n+K represents the total number of service nodes with K vehicles

T_{ij}^{m} = upper bound for time interval m for link (i, j)

t = the starting time from the depot (node 1)

b_k = weight (or volume) capacity of vehicle k

d_i = weight (or volume) to be collected at customer i; d_i = 0 for i = 1, n+1, …., n+K

B_1 = a large number

B_2 = a large number

B= $\max_k b_k$ = capacity of largest vehicle

L_i = earliest time that the vehicle can arrive at node i

U_i = latest time that the vehicle can arrive at node i

Decision Variables

$$x_{ij}^{m} = \begin{cases} 1 & \text{if any vehicle travels directly from node i to node j starting from i during} \\ & \text{time interval m} \\ 0 & \text{otherwise} \end{cases}$$

t_j = departure time of any vehicle from node j

w_j = weight (or volume) larger than or equal to that carried by a vehicle when departing from node j.

With this notation the TDVRP may be formulated as follows:

$$\text{Min} \sum_{k=1}^{K} t_{n+k} \qquad (2.16)$$

Subject to

$$\sum_{i=1}^{n} \sum_{m=1}^{M} x_{ij}^{m} = 1 \qquad (j = 2, \ldots, n+k) \qquad (2.17)$$

$$\sum_{j=2}^{n+k} \sum_{m=1}^{M} x_{ij}^{m} = 1 \qquad (i = 2, \ldots, n) \qquad (2.18)$$

$$\sum_{J=2}^{n} \sum_{m=1}^{M} x_{ij}^{m} = K \qquad (2.19)$$

$$t_1 = t \qquad (2.20)$$

$$t_j - t_i - B_1 x_{ij}^{m} \geq c_{ij}^{m} + c_j - B_1 \; (i = 1, \ldots, n; j = 2, \ldots, n+K; i \neq j; m = 1, \ldots, M) \qquad (2.21)$$

$$t_i + B_2 \; x_{ij}^{m} \leq T_{ij}^{m} + B_2 \qquad (2.22)$$

$$t_i - T_{ij}^{m-1} x_{ij}^{m} \geq 0 \qquad (i=1, \ldots, n; j=2, \ldots, n+K; i \neq j; m=1, \ldots, M) \qquad (2.23)$$

$$L_i + c_i \leq t_i \leq U_i + c_i \qquad (i = 1 \ldots n+K) \qquad (2.24)$$

$$w_j - w_i - B \sum_{m=1}^{M} x_{ij}^{m} \geq d_j - B \quad (i = 1, \ldots, n; j = 2, \ldots, n+K; i \neq j) \qquad (2.25)$$

$$w_1 = 0 \qquad (2.26)$$

$$w_{n+k} \leq b_k \quad (k = 1, \ldots, K) \qquad (2.27)$$

$x_{ij}^{m} = 0$ or 1 for all i, j, m (2.28)

$t_i \geq 0$ (2.29)

$w_i \geq 0$ for all i (2.30)

The objective function (2.16) minimizes the total route time of all vehicles (includes travel time, service time, and waiting time at all nodes). Constraints (2.17) to (2.19) ensure that each customer is visited exactly once and exactly K vehicles are used. Constraint (2.20) sets the starting time at node equals to some constant that remains same for all vehicles. If this constraint is omitted, the starting time of all vehicles will be the same but it will be a variable determined by optimization. Constraint (2.21) computes departure time at node j. The temporal constraints (2.22) and (2.23) ensure that the proper time interval m is chosen between nodes i and j according to the departure time from node i. Constraints (2.24) compel the time windows that are defined in terms of the arrival times at the nodes while the variables t_i for i = 1, 2, ….., n+k represent the departure times from the nodes. Constraints (2.25) to (2.27) inflict the capacity restrictions. Constraints (2.28) state that all vehicles leave the depot node empty. Constraints (2.25) ensure that the weight carried by the vehicle leaving customer j is at least equal to the weight when leaving the previously visited customer i plus the weight of the commodity picked up at customer j. Constraints (2.27) ensure that the capacity of each vehicle is not exceeded.

In TDVRP time taken to traverse node (i, j) is different from traversing the node (j, i). Nearest Neighborhood Technique, Sequential Route Construction Heuristic, Simultaneous Route Construction TDVRP Heuristic, Cutting plane heuristic were use to

solve the problem. The results obtained from each of the method were compared using various criteria. The formulation does not require subtour elimination constraints because both constraints (2.20) and (2.25) operate as SE (subtour elimination) constraints.

Most of the vehicle routing models assume travel times between the arcs as constant or they try to adjust the variation in travel times by some multiplier factors. Unfortunately, these assumptions may provide with suboptimal or sometimes infeasible solutions. Ichoua Soumia et al. (2001) have considered travel time to traverse an arc as a continuous function of the time of the day where speed is a step function depending on traffic during the day. In their model they have not assumed constant speed to traverse the arc but instead speed changes as it crosses the boundary between the consecutive time intervals. The procedure to calculate the travel time between node *i* and *j* is explained below:

The vehicle leaves node i at $t_o \in [\, t_k, \bar{t}_k \,]$ and the link (*i*, *j*) belongs to the subset of arcs. To limit the number of speed values v_{ijT} to estimate, the set of arcs A is partitioned into subsets $(A_c)_{1 \le c \le C}$. That is, the travel speed during period T on an arc (*i*, *j*) that belongs to a subset (or category) A_c is $v_{ijT} = v_{cT}$, where v_{cT} is the travel speed associated with category A_c and time period *T*. Variable d_{ij} is the distance between between the two nodes *i* and *j*, *t* denotes the current time and *t'* denotes the arrival time.

Procedure:

1. Set $t \leftarrow t_o$
2. Set $d \leftarrow d_{ij}$
3. Set $t' \leftarrow t + (d / v_{cT_k})$
4. while ($t' > \bar{t}_k$) do

4.1. $d \leftarrow d - (v_{cT_k})(\bar{t}_k - t)$

4.2. $t \leftarrow \bar{t}_k$,

4.3. $t` \leftarrow t + (d / v_{cT_k})$

4.4. $k \leftarrow k+1$

5. return $(t` - t_o)$

Tabu search algorithm is implemented to obtain the optimal result for the problem.

Donati, Montemanni, Casagrande, Rizzoli, Gambardella (2006) have considered the variable traffic condition in the vehicle routing problem to perform realistic optimization. Time Dependent Vehicle Routing Problem (TDVRP) consists of finding optimal routes by considering the time it takes to traverse each given arc depending on the time of the day travel starts. Assumptions for the problem are:

1) The Delivery/ Pickup to/from a customer location should be done in single delivery.
2) Total load to be delivered /pickup during a tour should not exceed the truck capacity.
3) All tours must start and end at the depot.

The total distance between the nodes is expressed in terms of the time taken to traverse the arc. Speed of the travel is inversely proportional to the time taken to traverse the distance. A step function is used for the speed distribution, from which the travel time distribution is derived. Multi-ant colony optimization is used to solve VRP with hard time windows. This approach consists of two artificial ant colonies. Each of the colony deals with one of the objectives of optimization: ACS-VEI (Ant Colony System) finds the minimum number of tours and ACS-TIME colony minimizes the total distance traveled.

In Time Dependent MAC-VRP, (Multi Ant Colony System) information regarding the time required to traverse the arc is deduced at the starting of the trip (i.e. if we know the departure time of the vehicle, we can predict the arrival time of the vehicle on the next node depending on traffic conditions). The arrival time at the destination node is calculated based on the travel time distribution depending on the speed of travel. Neighborhood sets were created around the customers to improve the speed and efficiency of the algorithm. Various local search techniques such as customer relocation, customer exchange, 2-k opt, branch relocation, branch exchange, post insertion and shuffling the tours order were used to improve the efficiency of the algorithm.

2.4 Simulated Annealing (SA)

The simulated annealing heuristic is based on the analogy with the thermodynamic annealing process. Thermodynamic annealing is a method in metallurgy to reduce the defects in a metal by heating metal to a very high temperature and then having a controlled cooling. This causes molecules with high-energy state to move randomly in their neighborhood to find a configuration with lower energy state than the current energy state. The result of this process is to have an ordered crystalline structure. If the metal is cooled too slowly or too fast, defects will be formed in the metal, which in our case represents local minima or maxima. Similarly, in simulated annealing (SA) heuristic it is important to have a proper cooling schedule. Initially the temperature is high in order to permit uphill moves and then it is lowered in a controlled manner till no possible improvement is possible. According to S. Kirkpatrick et al. (1983) the Boltzman

Law is used to determine the probability of acceptance of change in energy ΔE at the current temperature T and it is given by

$$p = \exp\left(\frac{\Delta E}{k_b T}\right)$$

where k_b is the Boltzman constant.

Uphill moves are accepted based on probability p. $\Delta E = E_2 - E_1$ represents change in objective function value where E_2 represents the function that gives total time of current solution, E_1 is a function that gives total time of the new solution generated and T represents control parameter in the heuristic. This probability is compared with a randomly generated number between [0, 1). Higher energy state solutions are accepted if the calculated probability is higher than the random number generated. From the current configuration, a new configuration is generated randomly through a transition mechanism that is in the neighborhood of the current solution. If the new configuration generated is associated with an objective function value E_1 that is lower than the current objective function value E_2, then the new solution is accepted and it becomes the current configuration. Otherwise, E_1 is accepted with some probability, $p = \exp\left(\frac{E_2 - E_1}{T}\right)$. This ability to change configuration enables simulated annealing to jump from local maxima or minima where most of the greedy algorithms get stuck. When the control parameter is high, the algorithm behaves more erratically searching different neighborhood to find global minima. When the control parameter is low, it behaves more as a descent search algorithm and tries to find the local minima. The parameters that are essential in successful implementation of SA are:

- Initial value of the control parameter

- Cooling schedule: Reduction in control parameter at each stage
- Transition Mechanism
- Equilibrium condition to reach steady state at each stage
- Stopping rule

Cerny (1985) proposed an analogy of statistical thermodynamics with simulated annealing and illustrated its application in solving TSP. He proposes to perform random permutations to obtain different neighborhood of the Traveling Salesman Problem. The probability of acceptance is given by Boltzmann Gibbs distribution described above. The results obtained were very close to optimal solution or even sometimes the optimal solution was obtained.

Eglese (1990) stated various modifications such as storing the best solution, sampling the neighborhood without replacement, and alternative acceptance probabilities for SA algorithm to improve its efficiency. Moreover, better solutions can be obtained by combining SA with some other approach. SA can either be used in the improvement phase to obtain a better solution or an initial solution can be found using SA and various methods such as branch and bound can be used to obtain a better solution.

Connolly (1990) applied simulated annealing to the Quadratic Assignment Problem. In the paper, the author has developed an improved annealing scheme which gives better results for a given range of problems. He has described various methods to obtain starting temperature, ending temperature and the cooling schedule and compared the results by applying it to various problem sets.

Breedam (1994) used simulated annealing based route improvement procedures to obtain a better solution. Route improvement procedures can be described as improvement within the tour (i.e, relocation of nodes within a tour) and improvement between the tours

(exchanging of nodes between different tours). He compared his algorithm with different search procedures and concluded that SA yields better solutions than other metaheuristic methods.

CHAPTER 3

MATHEMATICAL MODEL

In this chapter, we introduce a mathematical model for the single truck routing and driver scheduling problem with time-dependent travel, Hours of Service (HOS) regulations, multiple delivery locations and service time windows.

3.1 Problem Specification and Model Construction

In this thesis, we consider a truck with a full load that delivers goods to various customer locations. Each customer location l is associated with time window $[a_l, b_l]$. Each pair of customer locations (k, l) is connected through the highway network. There are many potential routes the truck can take on the transportation network as it departs from customer k to visit customer l next. Each segment on the highway network having a given speed profile is represented through a link (i, j). The arrival time at a customer location depends on the departure time from the preceding customer location. The time of departure from the origin depot is known. If the arrival at a customer location is before the start of the time window, the driver has to wait for the start of the time window to begin service. Depending on the arrival time and the ending time of the service window, the driver may take a break during waiting.

We have designed our algorithm by taking into consideration traffic congestion and HOS regulations enforced by the U.S. Department of Transportation for the safety of the truck drivers. Consideration of HOS regulations and traffic congestion add additional times (i.e., off–duty time and traffic delays) to reach a customer location. It has become

important to consider these practical constraints to schedule routes in order to minimize the total cost or time of the tour. The absence of these constraints may lead to sub-optimal or even sometimes infeasible routes.

3.1.1 Time Dependent Travel

Representing the transportation network, consider a graph G (V, E) where V is the set of vertices and E is the set of arcs. Each link (i, j) is assumed to have a fixed vehicle speed for each time interval. Let c_{ij}^{h} be the speed of the vehicle during the hth hour (i.e, h = 0, 1, 2, 3…, 23) of the day for a particular link (i, j). Speed distribution $c(t)$ is assumed to be a step function of the time (t) of the day, represented as follows: $c(t) = c_{ij}^{h}$, $h \leq t \leq h+1$. For example, link (i, j) has a variable speed distribution, i.e., a vehicle should have a speed of 40 miles/hr between 7:00 – 8:00 AM and 50 miles/hr between 8:00 – 9:00 AM. Similarly, different links have different speed distributions.

For simplicity, in this thesis, we consider a speed distribution with 24 intervals where each interval corresponds to an individual hour of the day. For more accuracy, we can model the time distribution with half an hour intervals or even with not on the hour break points. The travel time distribution on a link (i, j), is a continuous function of the time of departure from node i and it is derived from the speed distribution. Fig. 3.1 shows an arbitary speed distribution. Let the length of arc (i, j) be 2 miles. The time to reach node j depends on the time of departure from node i and the break time/ rest time that is added to the travel time if HOS regulations are observed. Suppose that the driver departs from the depot at 5:00 AM. Referring to the speed distribution, the speed corresponding

to 5:00 AM is 1.2 miles/ min. Travel time can be calculated using the formula *Travel Time = Distance/ Speed*. As the length of the arc is 2, the travel time to traverse the arc is (2/1.2) that is equal to 1.667 mins, as shown in the travel time distribution of Figure 3.2. If the time of departure is 6:00 AM, the driver will travel one hour with the speed of 1.2 miles/min and the rest of the distance with the speed of 0.6 miles/min. Distance travelled in one hour is (1) (1.2) = 1.2 miles. The remaining 0.8 miles will be travelled at speed of 0.6 miles/min. Therefore, total travel time is equal to 2.33 mins (i.e., 1 + (0.8/0.6)), as shown in the travel time distribution. The travel time increases as speed to traverse the arc decreases. Similarly, travel time decreases as speed to traverse the arc increases.

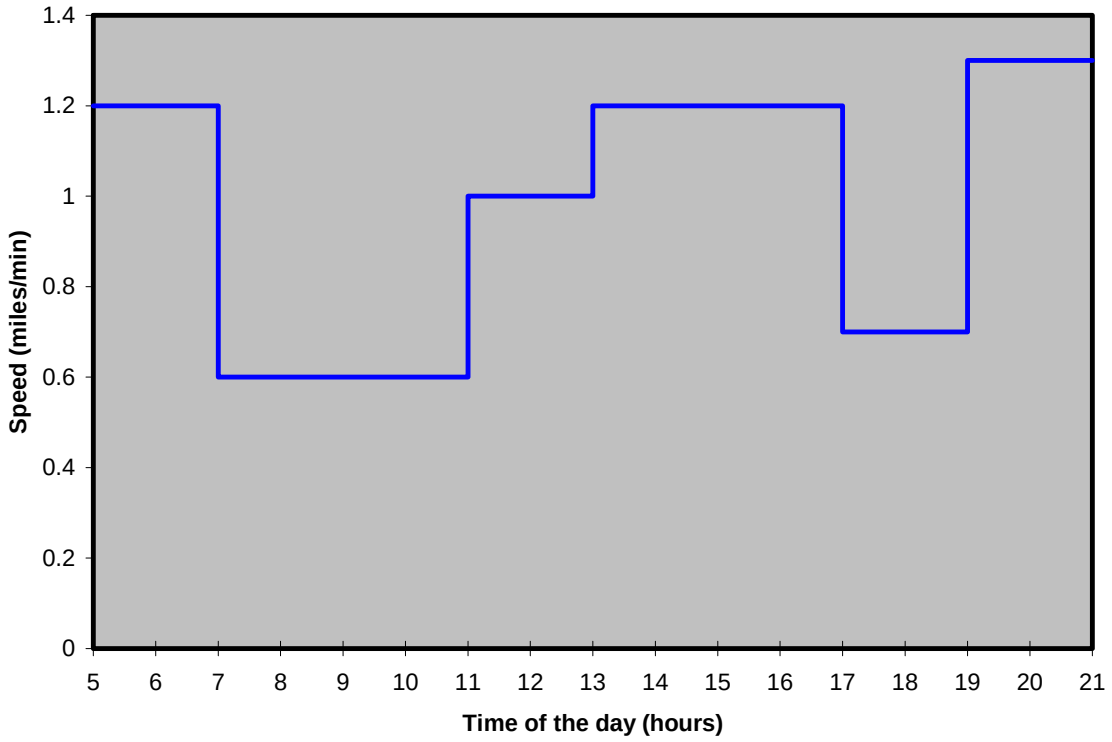

Fig 3.1 Speed Distribution

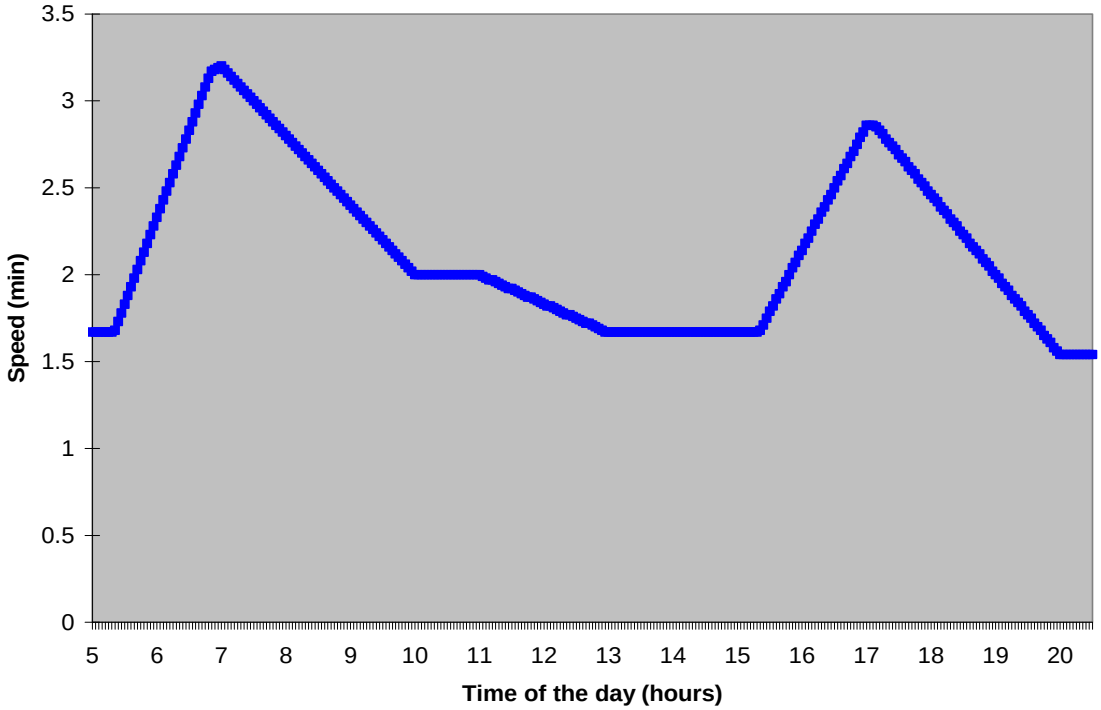

Fig 3.2 Travel Time Distribution

3.1.2 Mathematical Model

The truck starts its tour at the depot, visits all customers, and returns back to the depot. Each customer is associated with a node on the transportation network. Although it may be the same geographical location, the depot is associated with two nodes, the origin depot (k = 0) and the destination depot (k = n+1). The objective function here is to minimize the total time to complete a tour. Starting at node 0, visiting n customer nodes and terminating at node n+1. The total time is the sum of travel time, waiting time and service time at the customer node, and break time according to HOS regulations.

Before formulating the mathematical model, we define model parameters and decision variables below.

Sets:

G (V, E) = Graph representing the transportation network with a set of nodes V and set of Edges E

A= set of customer nodes, A={1, ….., n}, $A \subset$V

P_{kl} = set of paths from node $k \in A$ to node $l \in A$, $k \neq l$ on G (V, E)

P_{kl}^{r} = the rth path from customer node k to customer node l, $P_{kl}^{r} \in P_{kl}$

P_{kl}^{*} = shortest path from customer node k to customer node l

Parameters:

S_i = state of the driver at node i (i.e., accumulated driving time after most recent break, accumulated duty time after most recent break, accumulated weekly duty time after most recent weekly break)

$t_{ij}(t_i, S_i)$ = travel time on arc (i, j), when vehicle leaves node i at time t_i with driver at state S_i

$t_{P_{kl}^{*}}(t_k, S_k)$ = shortest time to reach customer node l from customer node k, when vehicle leaves node k at time t_k with the driver at state S_k

s_l = service time at customer node l (s_l =0, for l = 0 and n+1)

t_0 = the starting time from the origin depot (node 1)

M = a large number

a_l = earliest time that the vehicle can start service at node l

b_l = latest time that the vehicle can start service at node l

Decision Variables:

$$y_{kl} = \begin{cases} 1, \text{if vehicle visits customer } l \text{ immediately after customer } k \\ 0, \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, \text{if vehicle traverses arc } (i, j) \text{ of the transportation network} \\ 0, \text{otherwise} \end{cases}$$

t_j = departure time of vehicle from node $j \in V$.

The starting time of the trip from node 0, t_0 is given, as well as the state of the driver,

$S_1 = \{0, 0, 0\}$

Mathematical formulation:

$$\text{Min } t_{n+1} - t_0 \tag{3.1}$$

Subject to

$$t_{P^*_{kl}}(t_k, S_k) = \min_{P^r_{kl} \in P_{kl}} \sum_{(i,j) \in P^r_{kl}} t_{ij}(t_i, S_i) x_{ij} \qquad k \in A \cup \{0\},\ l \in A \cup \{n+1\},\ k \neq l \tag{3.2}$$

$$t_l - t_k - M y_{kl} \geq t_{p^*_{kl}}(t_k, S_k) + s_l - M \qquad k \in A \cup \{0\},\ l \in A \cup \{n+1\},\ k \neq l \tag{3.3}$$

$$a_l \leq t_l - s_l \leq b_l \qquad l \in \mathrm{A} \tag{3.4}$$

$$\sum_{j \in V} x_{kj} = 1 \qquad k \in A \cup \{0\} \tag{3.5}$$

$$\sum_{j \in V} x_{jl} = 1 \qquad l \in A \cup \{n+1\} \tag{3.6}$$

$$\sum_{i \in V, i \neq j} x_{ij} - \sum_{k \in V, k \neq j} x_{jk} = 0 \qquad j \in V \tag{3.7}$$

$$t_j \geq 0 \qquad j \in \mathrm{A} \tag{3.8}$$

$$x_{ij} \in \{0,1\} \qquad (i, j) \in E \tag{3.9}$$

$$y_{kl} \in \{0,1\} \qquad k \in A \cup \{0\},\ l \in A \cup \{n+1\}, k \neq l \tag{3.10}$$

The objective function (3.1) is to minimize total tour time i.e., the sum of travel time, delays due to time dependency, break time, waiting time and service time to complete a tour. Given the departure time from node k and the state of the driver (t_k, S_k),

constraint (3.2) states that if customer l is visited after k, then the shortest path from node k to node l is selected. The total time to traverse that path is the sum of travel time along the shortest time path from customer node k to customer node l, including the break time if necessary. Although it is not explicitly specified in the formulation, driver's break time is included, if necessary. Constraint (3.3) determines the departure time from customer node l if visited after customer node k ($y_{kl} = 1$). It is the sum of arrival time at customer node l plus the service time at node l plus the slack which is break time/ waiting time. Note that when $y_{kl}=0$ the constraint is always satisfied. Constraint (3.4) ensures that the start of the service at customer node l should be within the time window. If the driver arrives at a customer location before the start of the time window, he/she has to wait till the beginning of the time window. Constraint (3.5) and (3.6) ensure that there is only one outgoing arc from customer node k and only one incoming arc into customer l respectively, or equivalently, a feasible tour should include all customer nodes. Constraint (3.7) describes the balance equations for all nodes j of the transportation network. Constraint (3.8) states that departure time from customer node j should be non negative. Constraint (3.9) and (3.10) are the binary constraints.

CHAPTER 4

SOLUTION PROCEDURE

In this chapter, we describe the solution procedure for the mathematical model introduced in the previous chapter. A Simulated Annealing metaheuristic is developed that consists of two primary phases: the initialization phase and the improvement phase. A Time Dependent Dijkstra's (TDD) algorithm is used to construct an initial tour in the initialization phase. The simulated Annealing heuristic is implemented to improve the solution.

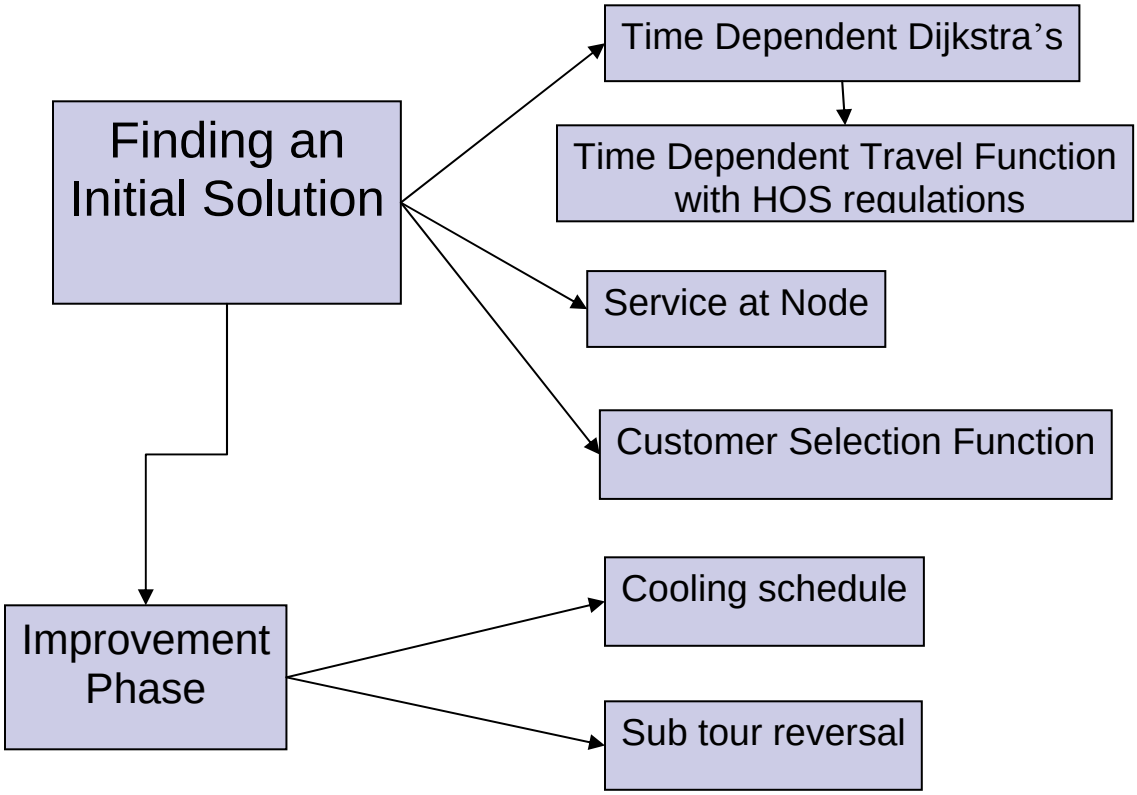

Fig 4.1 Overview of Solution Procedure

The two phases of the solution procedure are shown in Fig. 4.1. In the initialization phase, starting at the depot, we develop a greedy heuristic to find a tour to visit all customers and return to the depot. The greedy heuristic finds in each iteration the shortest time path to visit each customer l from the current customer (depot) k using the time dependent Dijkstra's algorithm. Since the travel time along each arc is not fixed, a time dependent travel function with HOS is incorporated into Dijkstra's algorithm to find the travel time along each arc (i, j). After the arrival time at each customer node l is found, the departure time is computed by adding service time, waiting time if arrival occurs outside the service time window of the customer, and break time if necessary. The next customer location (after k) to be visited is selected among the customers who have not been visited yet using the customer selection function. The above process is repeated until we find an initial tour. In the improvement phase, we use a simulated annealing schedule to improve the solution iteratively until certain halting criterion is met. The improvement phase returns a near optimal solution.

P = set of visited nodes

k = current customer node

l = customer node that has not been visited yet, $l \in A\text{-}P$

n = number of customers

Greedy Heuristic to find the initial tour:

$P = \{0\}$

$i = 0$

$k = 0$

while ($i \leq n$)

Find the shortest time to reach $l \in A- P$ from customer node k using TDD and find t_l, S_l, $l \in A- P$

Select customer node to visit next among $l \in A- P$ using the customer selection function

Let the node selected be k with (t_k, S_k)

$P = P \cup \{k\}$

$i = i+1$

end while

Complete the tour by connecting customer k with the destination depot.

4.1 Time Dependent Dijkstra's Algorithm (TDD)

Time Dependent Dijkstra's algorithm (TDD) is implemented to calculate the shortest time path P_{kl}^* between two customer nodes k and l (i.e., node k is the origin and node l is the destination) on a transportation network. In this Thesis, we have modified Dijkstra's algorithm by incorporating time dependent travel with HOS regulations.

The trip is initiated at the origin depot. The starting time of the trip is set to time t_0. The set of customers/depot nodes are connected through the highway arcs (set E) of the transportation network. Actually there are many paths connecting any two nodes k, $l \in A$ (set P_{kl}). In finding the shortest time path P_{kl}^*, the time dependent Dijakstra's algorithm employs the "Time Dependent Travel function with HOS regulations" that finds the time to traverse an arc (i, j). This is because the vehicle speed may be changing during the arc traversal or the driver may need a break. In the ordinary Dijakstra's algorithm the time to

traverse an arc (i, j) is considered constant. Given the departure time from node i, t_i and the state of the driver S_i= {*drt*, *dut*, *wdut*}, the function to calculate the arrival time at node j is shown in Figure 4.2 and the notation used is explained below. Assuming that the length of the trip does not exceed a week, we can simplify the notation by truncating the third component of the state of the driver, weekly duty time. The functions and the algorithm can be easily extended to the case where a trip may take more than week.

The notations for time dependent travel function with HOS regulations on arc (i, j) is as follows:

t: current time at node *i*

drt: accumulated driving time since last daily break

dut: accumulated duty time (driving time, service time and waiting time) since last daily break

wdut: accumulated weekly duty time since last weekly rest period

daily_max: maximum daily driving time between consecutive daily break periods (11 hours)

duty_max: maximum duty time between two consecutive duty break periods (14 hours)

weekly_max: maximum weekly duty time between two consecutive weekly rest periods (36 hours)

daily_break: time required for a daily break period (10 hours)

res_ddr: residual driving time before the next daily break period (i.e. *daily_max – drt*)

res_dut: residual duty time before the next daily break period (i.e. duty*_max – dut*)

res_wdr: residual weekly driving time before the weekly break period (i.e. *weekly_max – wdut*)

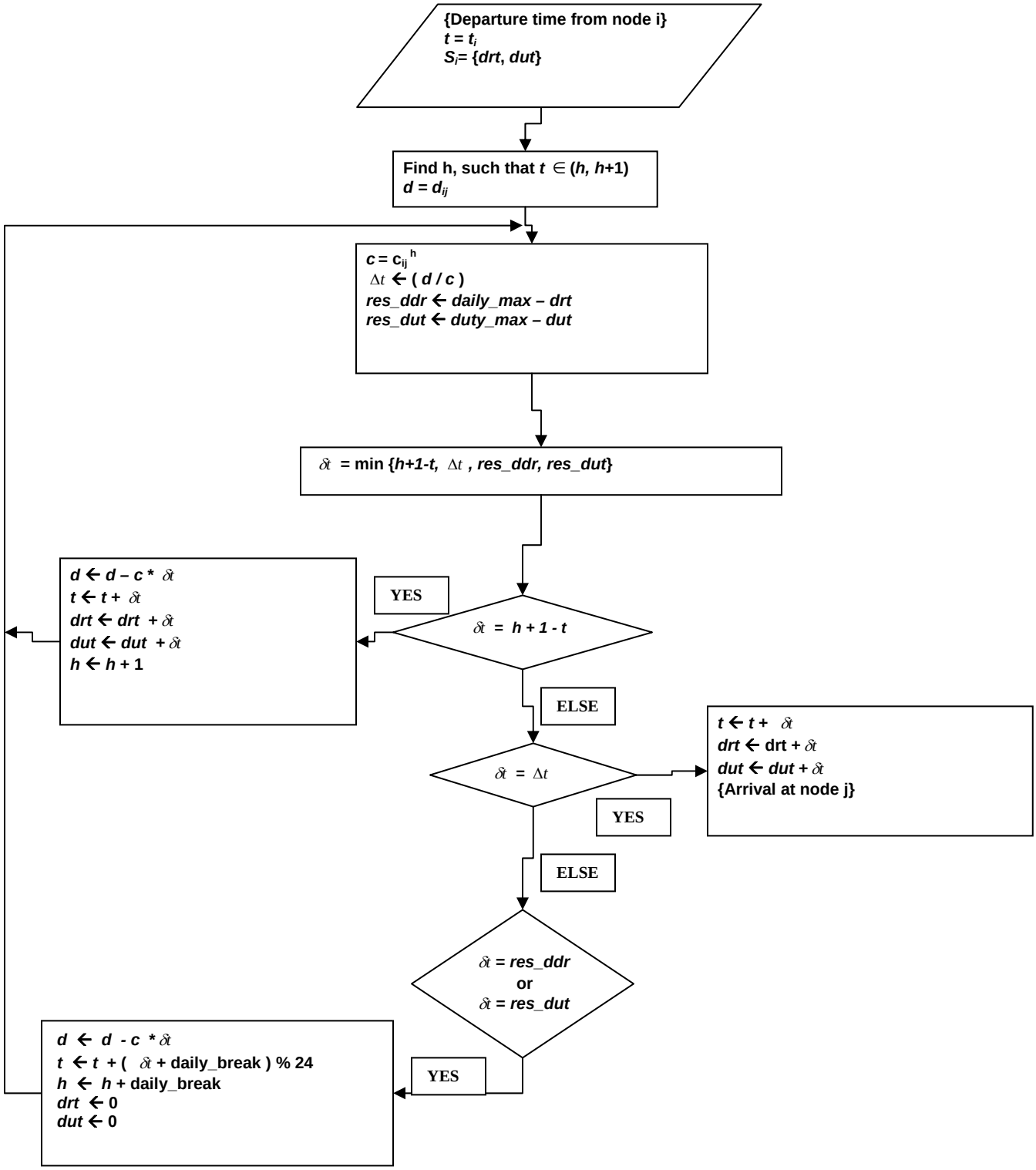

Fig 4.2: Time dependent travel function with HOS regulations

In the flowchart of fig. 4.2, the vehicle leaves node *i* at time *t*. The travel speed of the vehicle depends on the time of the day it departs from node i (i.e. v (t) = $c_{ij}^{h} \mid h \leq t \leq h+1$). The arrival time at node *j* depends on the speed of departure from node *i*. The state of the driver at node *i*, S_i is defined as a vector with two components, (1) the accumulated driving time and, (2) the accumulated duty time. The residual daily driving and, daily duty are computed next. Then, the earliest time δt is computed until one of the following events occurs: (a) speed on arc (*i*, *j*) changes, (b) node *j* is reached, (c) maximum daily driving time is reached, and (d) maximum duty time is reached. Whichever event occurs first, the current time and the state of the driver are updated. The recursive function is executed repetitively and the state of the driver is updated.

If the accumulated driving time or duty time is not sufficient to reach node *j*, the driver takes a daily break. Time of departure after taking the break is updated and daily driving (*drt*) & duty time (*dut*) are reset to zero. The speed to traverse the remaining distance depends on the time of the day the driver departs after taking the break. The above mentioned steps are repeated until node *j* is reached.

The shortest path P_{kl}^{*} from customer node $k \in A$ to customer node $l \in A\text{-}P$ is calculated using TDD. The procedure to calculate the arrival time at customer node *l* is explained below.

Pseudocode for TDD:

Dijkstra (Vertex *Start*)

for each vertex *v* in *Graph*: *// Initialization of timeto reach all the vertices in the graph to infinity*

Vertex *k* = *Start*

Vertex End = l

Initialize S*tart* vertex with (t_k, S_k)

Initialize *SettledNodes* to $\{k\}$ *// List of all nodes whose minimum time is obtained*

Initialize *UnsettledNodes* to $\{k\}$ *// List of all nodes directly connected to the SettledNodes whose minimum time is not obtained*

i* = *k

While *UnsettledNodes* **is not** empty:

for all arc $(i, j) \in E$:

Calculate total time to traverse the arc using time dependency function

$t_i + t_{(i, j)}$

if ($t_j > t_i + t_{(i, j)}$)

update :

time to reach node j

drt

dut

Add node j to *UnsettledNodes*

end for loop

Vertex i = arg (min t_j) overall the *UnsettledNodes* j *// Find the vertex with minimum Total time (i.e, summation of travel time, delays due to traffic congestion, break time)*

if (Vertex $i \in A$-P)

$P = P \cup \{i\}$

store (t_l, S_l)

if ($P = A$)

break

Add vertex i to *SettledNodes*

end while loop

return time and state of the driver for all unvisited nodes

Given the customer location k, time of departure t_k, and the state of the driver S_k, TDD finds the earliest time to reach an unvisited customer node l. All nodes j of the transportation network have their time to reach then (t_j) then set to infinity, except the starting node whose time is initiated at the departure time from node k. The state of the driver is initialized also to S_k. At the outset, when the trip is initiated at the depot, k=0, $t_k = t_0$ and $S_k = (0, 0)$. UnsettledNodes contains the list of nodes that are directly connected to the SettledNodes whose time to reach has not been obtained yet. SettledNodes contains the list of all vertices whose earliest time to reach has been obtained. Initially, UnsettledNodes and Settled Nodes are initialized with the origin depot. All the highway nodes j connected to current node i are stored in a adjacency list. For a given vertex i, the time to traverse arc (i, j) is calculated using Time Dependent Travel Time Function with HOS regulations, as described above. All of the adjacent nodes j connected to node i are added to the UnsettledNodes. The node j having the shortest time to travel form node i is found and stored in the set of settled nodes and we continue to follow the above procedure until we find the shortest time to reach all customer nodes $l \in A\text{-}P$. The state of the driver is updated at the customer node $l \in A\text{-}P$. At each iteration, where a partial tour up to customer node k having state (t_k, S_k) has been completed, this procedure is repeated to find the earliest time to reach a customer location which has not been visited yet, l.

4.1.2 Service at Customer Node

The arrival time at customer node l is calculated using TDD as t_k + $\min_{P_{kl}^r \in P_{kl}} \sum_{(i,j)\in P_{kl}^r} t_{ij}(t_i, S_i) x_{ij}$ where t_{ij} is the time to traverse arc (i, j) including break time in case the maximum of driving time or duty time has been reached during the traversal of arc (i, j). Each customer location has a time window [a_l, b_l] associated with it, within which the driver can start service, and the service time associated with the customer location is s_l. The earliest time a driver can start service is at the start of the time window (a_l). If the driver arrives before the start of the time window, he/she has to wait at the customer location or take a break before the service starts. Let w_{time} be the early arrival time interval i.e, the difference between starting time of the time window and vehicle arrival time.

Waiting time and service time at the node are considered part of the duty time. If after waiting and performing the service the driver does not exceed the maximum duty hours (14), he/she may wait and perform the service after which the state is updated to S_l^1. Otherwise, he/she can take a daily break and perform the service assuming that after taking the break, he/she does not miss the time window. If the latter happens, then he/she has to wait until the beginning of the time window on the next day to perform the service. On the other hand, the driver may take a break as long as the end of the break is before the right endpoint of the service window. In this case, the driver will be in an alternative state S_l^2 after he/she completes service at t_l. Both alternative states should be considered when node l is evaluated.

If the driver arrives within the time window, he/she may start service immediately. Service at the customer node should be performed without interruption. If the driver exceeds the maximum duty hours (14) while performing service, he/she should take a daily break before the start of service and perform service at the start of the time window on the next day.

If the driver arrives after the right endpoint of the time window, at customer location *l*, he/she may take the daily break and wait until the beginning of the time window the next day to start service. The departure time at the node is updated accordingly. Thus, the driver may have different courses of action depending on the arrival time at customer locations. These are explained in the flowchart of Fig. 4.3, illustrating the "Service at customer *l*" function.

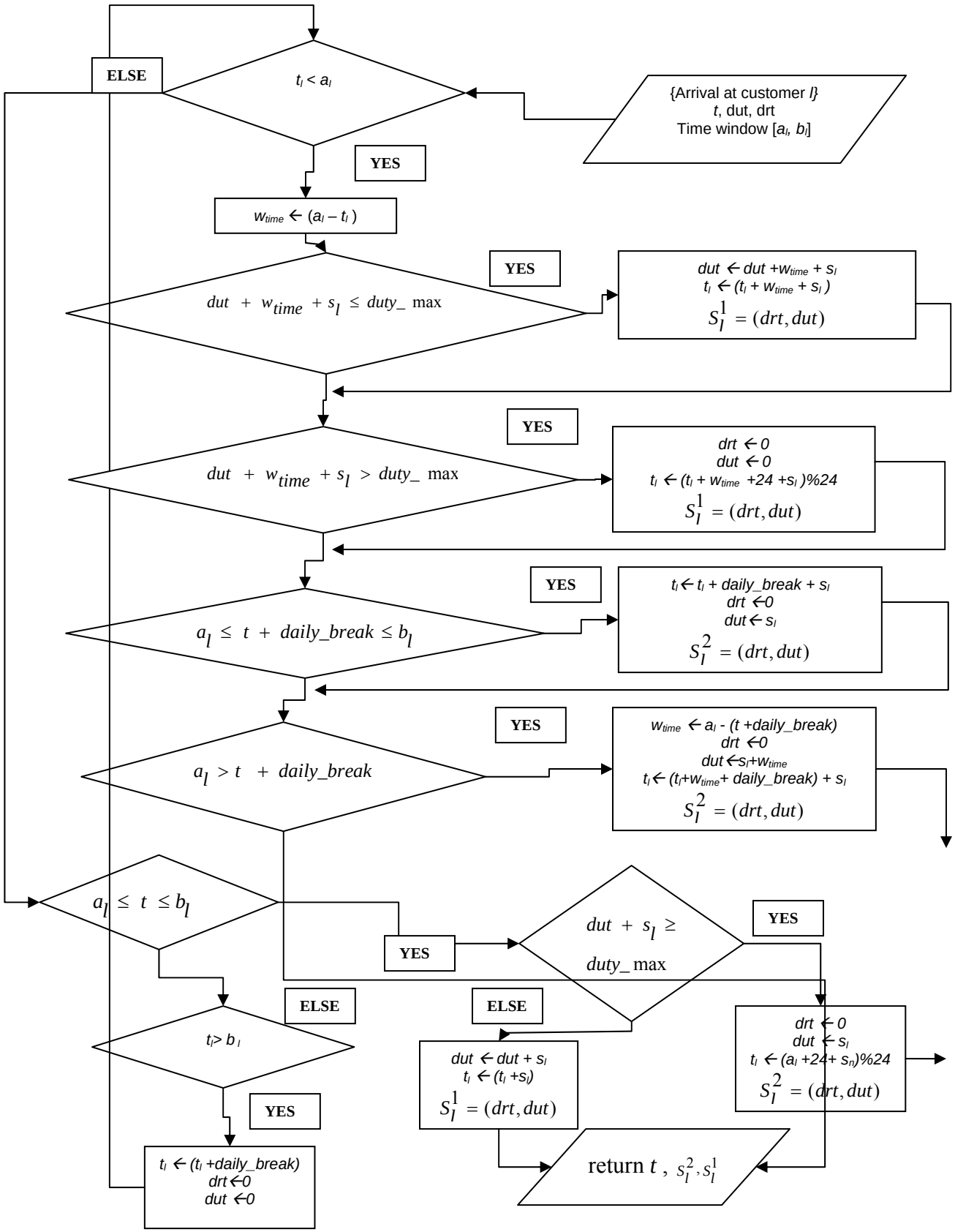

Fig 4.3: Service function at a node

The above function is used to determine the time of departure t_l from every remaining customer l, given the time of departure from the current customer k and the state of the driver S_k. The selection of the next customer node is made by the customer selection function. The customer selection function is the convex combination of two functions that compose the total time: operational time and non-operational time. The operational time (op_time) consists of driving and service times, thus time is directly related to the distribution operations. The non-operational time (non_op_time) consists of waiting time and break time, thus necessary times that indirectly support the distribution operations and HOS regulations.

Customer Selection Function = α * (op_time) + (1- α) *(non_op_time)

We evaluate this function for a given value of α, $0 \leq \alpha \leq 1$. When α is less than 0.5 we show preference in reducing waiting time at the customer and avoising break time. When α is greater than 0.5 we show preference in reducing driving duty time. By repeating the initialization process with a different value α, a new initial solution is generated. Thus, the function allows for controlling the diversity of initial solutions. The best solution can be used for the starting solutions in the SA heuristic.

4.2 Solution Improvement with Simulated Annealing

Simulated Annealing generates solutions in the neighborhood of the current solution and evaluates them. A random generation method is used to select two nodes from the current solution. The two nodes are swapped to obtain a new solution. According to Connolly (1990), this approach might be inefficient as potential

improvements might be missed at lower temperatures and secondly escaping the local minima before it is reached due to the random nature of selection of nodes. Sequential search method can be used but this method is not suitable to all kind of problems especially those problems where natural ordering causes search to perform badly. Thus the quality of the solution depends on the selection of node. Various features used to improve efficiency of Simulated Annealing are described next:

1) Initial temperature

A generic approach based on Connolly's strategy (1990) is used in our experiments. We performed M = 50 k (where k = 0.5 (n) (n-1)) random swaps initially to find δ_{max} (Largest uphill step) and δ_{min} (smallest uphill step). The intial temperature T_0 is set to δ_{max} (thereby giving an initial probability greater than 0.4). It is computes using the formula

p = e^ ((evaluation (current solution) - evaluation (new solution))/T_0

2) Cooling Schedule

It controls the rate at which the temperature changes and helps the heuristic to escape local minima. It causes the heuristic to act more erratically when the temperature is high and as a result of that, at higher temperatures the probability of acceptance of worse solution is much more than that at a lower temperature. The cooling schedule can be linear, exponential or polynomial. Selecting a schedule is the trade-off between the quality of solution and the computational time to find the solution. A proper cooling schedule can be determined by trial and error. If we decrease the temperature too quickly, the system will get "quenched", that is the system is still in the higher energy state (higher objective function value) and the temperature is too low to find a tour with lower energy state (lower objective function value). As a result of it, we get stuck into local minima and the algorithm would not be able to find a good solution.

Each temperature t_k, after reaching the equilibrium, we reduce the temperature by γ. Thus $t_{k+1} = \gamma * t_k$. The range of γ is between [0.5, 0.9]. We have done experiments with different cooling schedules and compared the quality of the solutions obtained using different schedules. In order to find good final solutions, very close to the optimal, we used $\gamma = 0.9$ in the computational experiments.

3) Transition Mechanism

We can arrange the n customer in n! permutations. By the way, a reverse order in a permutation yields a different solution because in our case distances (times) are not symmetric due to time dependent travel. It is possible to generate all permutations of given n-1 customer nodes within finite time. However, the computational time of complete enumeration of solutions is increasing exponentially with *n*. Thus we implement heuristic method to obtain approximate solution in reasonable amount of time. A sub tour reversal (i.e. exchanging of two nodes in a given tour) generates new tours. The new tour generated is evaluated and total time to complete the tour is compared with the previous tour. Sub tour reversal requires selecting a beginning slot and ending slot from a given sequence of the cities. The beginning slot and ending slot can be anywhere except from first and the last slot from the given sequence. Random number generation is use to select beginning and ending slot. The time of departure in the new tour from all nodes preceding the beginning slot remains unchanged. The departure times from the remaining nodes have to be recalculated using the TDD algorithm.

4) Equilibrium Condition for reaching a steady state at each temperature

A number of iterations should be performed at each temperature in order to reach equilibrium condition. In our experiments, we used Burkard and Rendl's (1984) who suggestion for the TSP problem to perform, $L = 0.5 * n^2$ (where n is the number of

customers) iterations at each stage in order to reach equilibrium. Moreover, authors have suggested multiplying L with 1.1 to perform more iteration at lower temperatures in order to find better solutions. By experimentation we found that multiplicative factor of 1.03 produces equally good results as compared to results obtained using multiplicative factor of 1.1. By decreasing multiplicative factor decreases computational time considerably.

5) Stopping Rule

The heuristic will stop once the halting criterion (minimum temperature) is reached. We set halting criteria as $\delta_{min.}$ The heuristic can be stopped earlier after certain number of iterations (approximately after 3200 iterations), if the cumulative difference of the best solution known so far for the last 5 consecutive solution is less than or equal to minimum reduction rate (ε). From the experimentation, we can decide upon the number of iterations to be performed, after which cumulative difference of the last five iterations is recorded. In our experiments, either minimum reduction rate or halting criteria, whichever occurs first, act as the stopping criterion.

Pseudocode for SA (for minimization)

```
Procedure simulated annealing
 Begin
 Current solution
 best solution ← current solution
 initialize temperature T
 initialize halting criteria
 while (temperature > halting criteria)
        for i:= 1 to50 do
        select randomly two nodes from current solution and swap them
```

```
        Evaluate new solution
        if evaluation(new solution) < evaluation(current solution)
                current solution ← new solution
        else if random[0,1) < e^((evaluation(current solution)-evaluation(new
                solution))/T)
                current solution ← new solution
        if evaluation(current solution) < evaluation(best solution)
                best solution ← current solution
        end for loop
    temperature ← schedule(temperature)
    i = integer part of (50*1.03)
    if ( cumulative difference of the last four stages < 0.01)
        break
end
```

Simulated annealing starts with an initial solution. At each stage *k*, a number of iterations are performed to attain equilibrium. The new configuration is generated by randomly selecting two nodes and swaping them. The change in the total travel time is ecomputed. The evaluation of the trip is done starting from the customer where the change has taken place. The new solution is always accepted if the objective value E_1 is less than the objective function value of the current solution E_2. The current configuration then becomes the new configuration. If the objective function value $E_1 > E_2$, the new solution is accepted based on the probability $p = \exp\left(\frac{E_2 - E_1}{T}\right)$. This probability calculated is then compared with a random number generated with a uniform probability

distribution in [0, 1). If r < p = exp $\left(\frac{E_2 - E_1}{T}\right)$ then the new solution is accepted. The current solution is updated to the new solution. If the random *number* generated is greater than the probability, we do not update current solution. Once the equilibrium is reach, temperature is lowered by γ to T.

CHAPTER 5

Computational Results

In this chapter, we compare the optimal solutions found by exhaustive enumeration to the solutions obtained by the Simulated Annealing approach (heuristic) for small size problems. The problems were generated by considering major cities in the Northeast U.S. as the customer locations. Distances of highway segments connecting customer nodes were derived with google maps (refer to appendix C for the data). The comparison is performed for small size problems (i.e., 4, 5, 6 customer nodes). Results for bigger problems consisting of 9, 14, 19, 24, 29 customer nodes and a depot have been evaluated on the basis of the computational time required to get the final solution.

5.1 Small Problem Size Comparison

The exact optimal solution was found by evaluating all possible tours which is n! in number. This is twice the number of the TSP tours because due to asymmetric distances a tour in the opposite directions has different total time. For every tour the, TDD function with HOS regulations was used to find the trip time. The accuracy of the heuristic has been tested on different initial solutions obtained by different values of α. Values of α used in the experiments are 0.3, 0.5, 0.7, 0.9 and 1. A cooling schedule with $\alpha = 0.9$ is used in all experiments. The initial temperature is obtained by running $M = 50$ K (where $K = 0.5\ (n)\ (n-1)$) experiments. The largest uphill step found in the iterations is set as the initial temperature and the lowest uphill step found is set as the halting criterion. $L = 0.5\ n^2$ iterations performed at each stage (temperature). Both exhaustive

enumeration and SA heuristic are coded in JAVA using Eclipse 3.4. The Enumeration code is presented in Appendix A and the SA Heuristic code is presented in Appendix B. The computer programs were run on an Intel processor 2.0 GHz. The cities included in our experiments with their abbrevations:

Columbus, OH (COL), Pittsburgh, PA (PIT), New York City, NY (NYC), Washington DC, (WDC), Philadelphia, PA (PHI), Charlotte, NC (CHA), and Richmond, VA (RIC)

Problem specifications and comparison results are given in Table 5.1. The following abbreviations are used:

EN: Enumeration Method

SA : Simulated Annealing Method

NC : Number of Customers

CT : Computation Time (secs)

TT_{en} : Total time for tour using enumeration method (secs)

TT_{sa} : Total time for tour using simulated annnealing method (secs)

Table 5.1. Comparison of enumeration and SA results

		EN		SA		
	NC	**CT**	**TT_{en}**	**CT**	**TT_{sa}**	Cities Included
1	4	4	78.48	6	78.48	COL, PIT, NYC, WDC,PHI
2				5	78.48	COL, PIT, NYC, WDC,PHI
3				5.2	78.48	COL, PIT, NYC, WDC,PHI
4				6.5	78.48	COL, PIT, NYC, WDC,PHI
5				6.5	78.48	COL, PIT, NYC, WDC,PHI
6	5	23	102.06	8	102.06	COL, PIT, NYC, WDC,PHI,CHA
7				9	102.06	COL, PIT, NYC, WDC,PHI,CHA
8				9	102.06	COL, PIT, NYC, WDC,PHI,CHA
9				13.9	102.06	COL, PIT, NYC, WDC,PHI,CHA
10				13.5	102.06	COL, PIT, NYC, WDC,PHI,CHA
11	6	150	104.83	16.4	104.83	COL, PIT, NYC, WDC,PHI,CHA, RIC
12				16.5	104.83	COL, PIT, NYC, WDC,PHI,CHA, RIC
13				17	104.83	COL, PIT, NYC, WDC,PHI,CHA, RIC
14				17	104.83	COL, PIT, NYC, WDC,PHI,CHA, RIC
15				16.6	104.83	COL, PIT, NYC, WDC,PHI,CHA, RIC

As Table 5.1 shows, the SA heuristic consistently produces the optimal solution as compared to the result obtained through the enumeration method in all the three experiments. For the four-customer problems, including depot, both SA and Enumeration solved the problems in about the same time (4-6.5 secs). For the six-customer problems, computation time is 150 secs for enumeration method as compared to at most 17 seconds required by SA method.

The above comparison shows that for considerably small size problems (maximum six customers and a depot), SA provided with the optimal solution. Of course the problem with the enumeration method is that with the increase in the number of

customer nodes the computational time increases exponentially whereas SA is much more efficient in computation time.

In the next section, we discuss the quality of solution obtained by having different values of α and cooling schedules with nine customers and a depot. We tested our heuristic with 14, 19, 24 and 29 customers. We can include more cities depending on our requirements and can develop data for the network of highways for those cities using google maps.

5.2 Performance in Larger Size Problems

In order to illustrate our model and the solution approach in a real life transportation network, we have constructed a realistic example with nine customers located in major cities in the northeast U.S. and a depot at Columbus, OH. Appendix C contains the data for this problem. A map of the Northeast U.S is shown in Fig. 5.1 with the cities and the underlying highway network highlighted. As it was difficult to obtain real time traffic congestion data for each arc, we have assumed same speed distribution for all arcs. The model can be modified to real time traffic congestion upon availability of the data.

The computational time required to obtain the optimal solution using exhaustive enumeration for 10 nodes was more than a day while with SA it took on the average 7 minutes to find an optimal solution or near optimal solution. Different initial solutions (different starting points in the solution space) are obtained using different values of α, α = 0.3, 0.5, 0.7, 0.9. The objective value obtained by SA was compared to the optimal objective value obtained by the enumeration method.

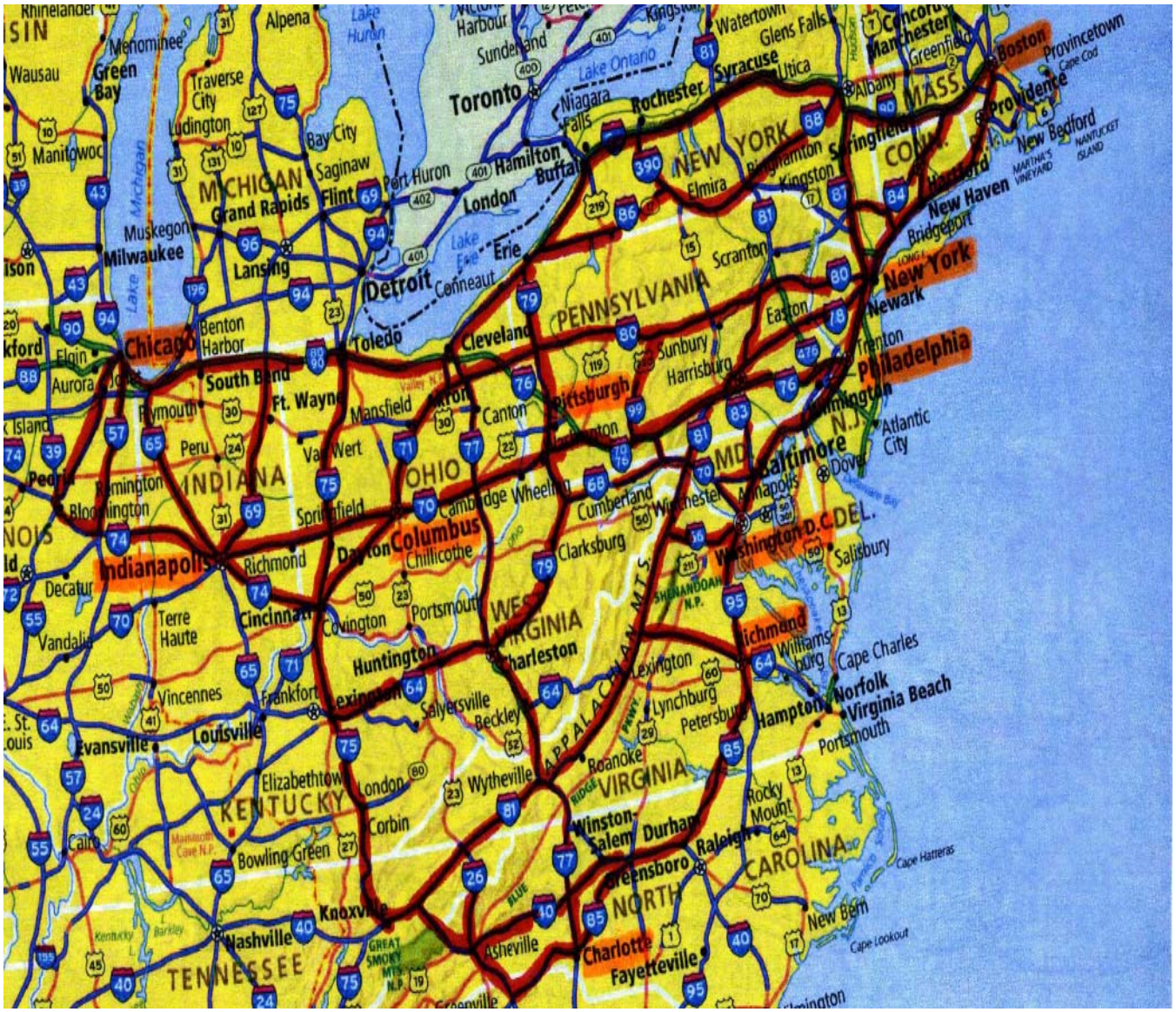

Fig 5.1 Transportation network of Northeast U.S.

Table 5.2. Comparison of results using different initial solution and cooling cchedule

Weight (α)	Initial objective value	Cooling schedule (γ)	Objective value of SA metaheuristic	Optimal Objective Value	Deviation from the optimal solution
0.3					
	217	0.7	135.874	135.874	0.00%
		0.8	135.874		0.00%
		0.85	140.22		3.20%
		0.9	135.874		0.00%
0.5					
	196	0.7	135.874	135.874	0.00%
		0.8	135.874		0.00%
		0.85	140.154		3.15%
		0.9	140.154		3.15%
0.7					
	140.22	0.7	135.874	135.874	0.00%
		0.8	135.874		0.00%
		0.85	135.874		0.00%
		0.9	135.874		0.00%
0.9					
	176.17	0.7	135.874	135.874	0.00%
		0.8	140.154		3.15%
		0.85	140.154		3.15%
		0.9	135.874		0.00%

With each initial solution obtained by a given value of α, 4 SA experiments were conducted using different cooling schedules. γ = 0.7, 0.8, 0.85, and 0.90 for a total of 16 experiments for all the combinations of values of α and γ. We were able to locate the optimal solution about 70% of the time. The second best feasible solution was found by SA when the optimal solution was not obtained (30% of the time). The deviation from the optimal objective value is below 1%. Moreover with α =0.7 and cooling schedule with γ equal to 0.7, 0.8, 0.85 and 0.9 we consistently obtained optimal solution. The results obtained by SA method mainly depend on the initial solution, initial temperature, number

of iterations performed at each stage and the cooling schedule. Slower cooling requires more computational time but gives better quality solutions. Thus, by varying the cooling schedule we trade-off the quality of solution and the computational time. By trial and error, we can determine the optimal cooling schedule for a problem having 10 customer nodes including the depot. The initial temperature is obtained by running M = 50 K (where K = 0.5 (n) (n-1)) experiments which is 4500 iterations. The largest uphill step found in the iterations is set as the initial temperature (δ_{max} = 180) and the lowest uphill step found is set as the halting criterion (δ_{min}=3). L = 0.5 n^2 iterations were performed at each stage (temperature).

To make the model more realistic, we considered time dependent arcs. If the arcs, were not time dependent the optimal solution would not have been changed by varying the departure time from the depot. To assess the impact of dependent arcs consideration in making accurate trip time predictions in the model, we varied the departure time from the depot in hour intervals during the course of a day and plotted the optimal trip times in Fig.5.2. It appears that a shorter trip time is obtained when the departure is 2 AM in the morning. The worst departure time appears to be around 12 AM. The difference between the longest and the shortest trip time in Fig. 5.2 is 38.22 hours, or 29% of the shortest trip time. Total time of travel decreases sharply when the vehicle departs from the depot right after the midnight and increases as the vehicle leaves later during morning hours and during afternoon. Traffic during the day causes delay to reach the customers and as a result of that, driver misses the time window to perform the service. He/ she has to wait to begin service on the next day. Since the waiting time at the customer nodes varies considerably due to traffic during peak hours and break time due to HOS regulations the sequence customers are visited changes. Thus the optimal solution changes as we change

the departure time during the day. Valleys shown in the graph below, is due to the reasons described above.

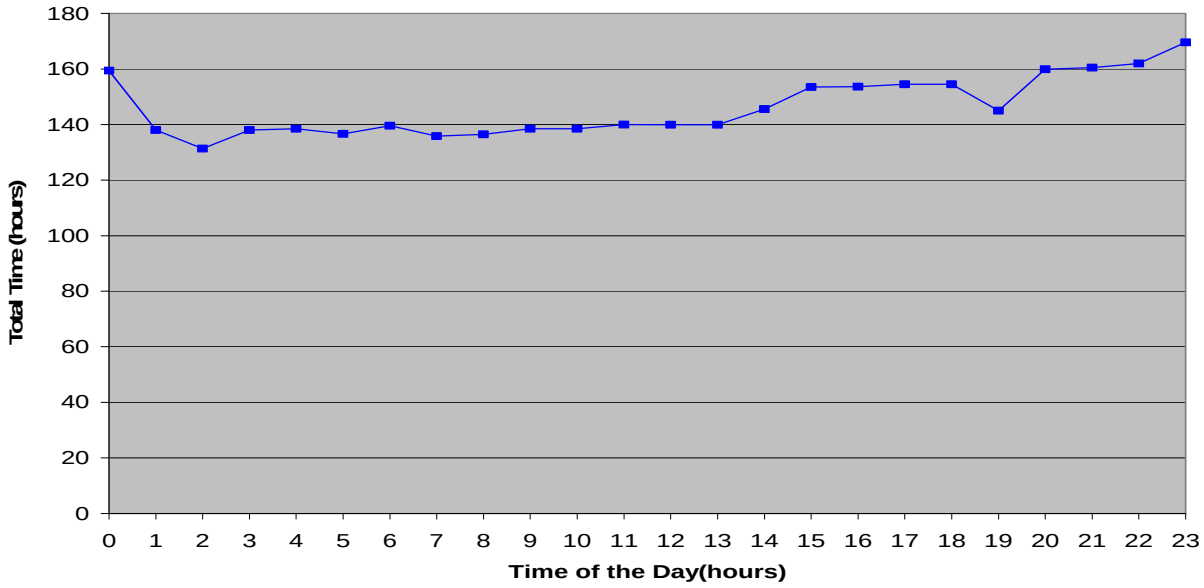

Fig 5.2. Travel time distribution

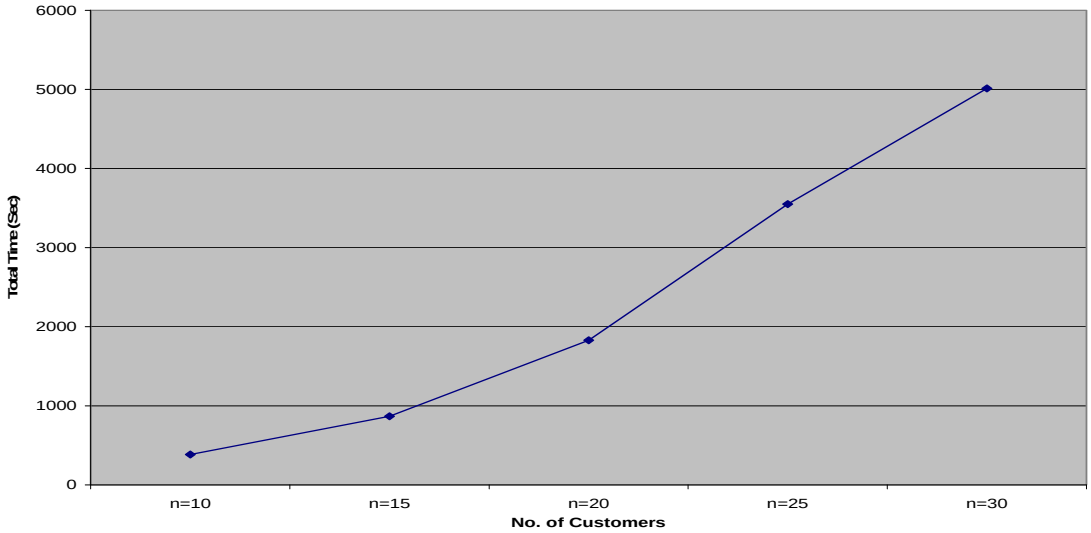

Fig 5.3: Computation time for larger size problems

In Figure 5.3, we have plotted the average computational time required to solve the problem with varying number of customers (including depot). The standard deviation with 10, 15, 20, 25 and 30 customers is 38.00, 50.77, 73.33, 104.86, 149.37 secs respectively. We have performed 10 iterations for each set of customers. All the experiments where performed with $\gamma = 0.9$ cooling schedule and $L = 0.5(n^2)$, (as suggested by Burkard and Rendl (1984)) number of iterations at each stage. The initial temperature is obtained by running M = 50 K (where K = 0.5 (n) (n-1)) iterations. The largest uphill step found in the iterations is set as the initial temperature (δ_{max}) and the lowest uphill step found is set as the halting criterion (δ_{min}). L = 0.5 (n^2) iterations were performed at each stage and the number of iterations performed were increased by multiplying L with 1.03 in successive stage (temperature). It is observed that computational time increases polynomially as the number of customers' increases.

In order to find the relationship between the computational time and the number of customers visited, we developed a univariate regression model, $t = b\ n^c$ where b and c are positive constants and n is the number of customers, excluding the depot. We used SPSS 17.0 to determine the parameters of the model. From the experiments the values of b and c came out to be 2.506 and 2.257, respectively, or $t = 2.506 * n^{2.257}$. The fit is good as indicated by the R^2 value (0.984).

CHAPTER 6

DISCUSSION AND CONCLUSIONS

In this Thesis, we have developed a time dependent truck routing and scheduling model with time windows and Hours of Service (HOS) regulations. We used a greedy heuristic to generate good initial solutions to the problem which were further improved by a Simulated Annealing (SA) metaheuristic.

In order to assess the quality of the solutions derived by the heuristic, we used exhaustive enumeration to find the optimal solution for small size problems. The enumeration method takes more than one hour to solve problems with 6 customers and more than a day for problems with 9 customers. The comparison shows that the heuristic approach provides near optimal solutions in reasonable amount of time. The computational efficiency and accuracy of the heuristic depends on the quality of the initial solution, equilibrium condition and the cooling schedule used to solve the problem. The optimal initial temperature and cooling schedule in the SA heuristic were determined by trial and error. We evaluated our results starting with different initial solutions and concluded that applying SA results in final solutions which are within 1 % of the optimal objective value and 75% of the time results in the optimal solution.

In the future, GIS could provide more accurate data on vehicle speeds and distances on the transportation network which could be incorporated into our algorithm for more accurate results. Moreover, our algorithm can be improved if it is equipped with some memory of the recent moves made to generate new solutions. This is because the same moves can be repeated again which results into reverting back to the same solution.

In order to avoid this problem, tabu search can be combined with simulated annealing to provide a near optimal solution in less computational time.

This routing problem can be further extended to more than one vehicle with pickup and delivery and more than one depot for scheduling near optimal routes covering all customer nodes and satisfying HOS regulations.

REFERENCES

1. Archetti C., and Savelsbergh M., (2007), "The trip scheduling problem", Working paper, Georgia Institute of Technology, Atlanta, GA, www.isye.gatech.edu/~mwps/publications/.
2. Baker E.K. (1983), "An exact algorithm for the time constrained traveling salesman problem", *Operations Research*, Vol. 31, pp. 939 – 945.
3. Bianco L., Mingozzi A., and Ricciardelli S. (1998),"Dynamic programming strategies for the traveling salesman problem with time windows and precedence constraints", *Operations Research,* Vol. 45, pp. 365 – 378 .
4. Breedam A.V. (1995), "Improvement Heuristics for the Vehicle Routing Problem based on Simulated Annealing", *European Journal of Operational Research*, Vol. 86, pp. 480-490.
5. Brooks S.P. and Morgan B.J.T (1995), "Optimization using Simulated Annealing", *Operations Research*, Vol. 44, pp. 241-257.
6. Burkard R.E., Rendl F. (1984), "A thermodynamically motivated simulation procedure for combinatorial optimization problems", *European Journal of Operational research*, Vol. 17, pp. 169-174.
7. Calvo R.W. (2000), "A new heuristic for traveling salesman problem with time windows", *Transportation Science*, Vol 34 , pp. 113-124.
8. Cerny V. (1985), "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *Journal of Optimization Theory and applications*, Vol. 1, pp. 41-52.
9. Clair L.A. and Fox S.D. (2004), "Time to Simplify Trucking Tariffs (Less Than Truckload)," *Supply Chain Management Review*, Vol 8, pp. 35-39.
10. Connolly D.T. (1990), "An improved annealing scheme for QAP", *European Journal of Operational Research*, Vol. 46, pp. 93-100.

11. Donati A., Montemanni R., Casagrande N., Rizzoli A., Gambardella L., (2006) "Time dependent vehicle routing problem with a multi ant colony system" *European Journal of Operations Research*, Vol. 185, pp. 1174- 1191.
12. Dumas Y., Desrosiers J. and Gelinas E., Solomon M. (1995), "An optimal algorithm for the traveling salesman problem with time windows", *Operations Research*, Vol.43, pp. 367-371.
13. Eglese R.W. (1990), "Simulated Annealing: A tool for Operations Research", *European Journal of Operational Research*, Vol. 46, pp. 271-281.
14. Federal Motor Carrier Safety Administration, 2005. Hours-of-Service Regulations, http://www.fmcsa.dot.gov/rules-regulations/topics/hos/hos-2005.htm.
15. Goel A., and Gruhn V., (2006) "Drivers' working hours in vehicle routing and scheduling", Working Paper, University of Leipzig, Leipzig, Germany http://ebus.informatik.unileipzig.de/papers/paperuploads/Drivers&%238217__working_hours_in_vehicle_routing_and_schedulingAsvin_Goel__Volker_Gruhn8096.pdf.
16. Hillier and Lieberman. Introduction to Operations Research 8th edition, McGraw Hill publishers, New York.
17. Osman H.I., (1993) "Metastrategy simulated annealing and tabu search algorithms for vehicle routing problems", *Annals of Operations research*, Vol. 41, pp. 421 – 451
18. Laurent B., Guihaire V., and Hao J., (2005) "A heuristic solution for a driver-vehicle scheduling problem, Operations Research Proceedings", Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Bremen, September 7–9.
19. Malandraki C., Daskin M., (1992) "Time dependent vehicle routing problems: Formulations, Properties, and Heuristic Algorithms" *Transportation Science*, Vol. 26, pp. 185-200.

20. Portugal R., Ramalhinko H., and Paixao J.P., (2006) "Driver Scheduling Problem Modeling", Working Paper, Universida de de Lisboa , Portugal http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1002565.
21. Rene V.V. Vidal (1993). Lecture Notes in Economics and Mathematical Systems: Applied simulated annealing, Springer- Verlag, Berlin.
22. Soumia I., Michel G., Jean-Yves P., (2003), "Vehicle dispatching with time-dependent travel times", *European Journal of Operational* Research, Vol. 144, pp. 379–396.
23. S Kirkpatrick, C.D. Gelatt Jr., M.P.Vecchi, (1983), "Optimization by Simulated Annealing", *Journal of Operations Research*, Vol. 220, pp. 671-680.
24. Timothy G., 2007, "An operating room scheduling problem solved using simulated annealing", MS Thesis, Dalhousie University, Canada.
25. V.P.J.M Laarhoven and E.H.L. Aarts (1987), Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, Norwell, Massachusetts.
26. WERC (2004), "Under the Gun: As the New Trucking Hours of Service rules take the hold, is the Warehousing Industry ready?" WERC Sheet, March, pp. 1-3.
27. Xu H., Chen Z.L., Rajagopal S., and Arunapuram S., (2003), "Solving a practical pickup and delivery problem", *Transportation Science*, Vol. 37, pp. 347-364.

APPENDIX A

Enumeration Code:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Deque;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.Random;
import java.util.Set;
import java.util.StringTokenizer;
import java.lang.Math;
import java.lang.reflect.Array;

// Represents an edge in the graph.
class Edge {
	public Vertex dest;
	public Vertex source;
	public float time;
	float nondr;
	float cost;
	float tdaily = 11;
	float tduty = 14;
	float tweekly = 60;
	float trest = 34;
	float tbreak = 10;
	float lweekly = 0;
	float ldaily = 0;
	float lduty = 0;

	public Edge(Vertex d, float c) {
		dest = d;
		time = c;
	}
```

```
        public Edge(Vertex s, Vertex d) {
                source = s;
                dest = d;

        }

public int[] boi84 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i84bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i84i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i91i84 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i91i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i91i9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nyi9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] boi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i95bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i95i9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195i95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] boi87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i87bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i87ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] nyi87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i87i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i8690i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i87i88 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i88i87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i88i86 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i86i88 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i86i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i86 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nyi8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8099i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i76oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ohi76 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i76 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i76i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7590i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i78i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i81i78 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i8176i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i99i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i70oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ohi70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i6970in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ini6970 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ohin = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] inoh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ini6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i6590in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i6590ch = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini57 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i57in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i57i94 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i94i57 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i94ch = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] chi94 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i6970 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6970i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6590i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i7590i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini275 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i275in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i275i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i75i275 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i75oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ohi75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i75i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i64i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i64i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i77i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i64i40 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i40i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i40i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i81i40 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chai8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81us74 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] us74i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] us74i85 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i85us74 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i85cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] chai85 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i75i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7590i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chachi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi95cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi95va = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vachi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8164i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8164va = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vai8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vawa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] wava = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i8164i8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166i8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166wa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wai8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wawai95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wai95wa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] wai95pa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] pawai95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176i8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176pa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] pai8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] panjt = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] njtpa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] njtny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] nynjt = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i70i376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i376i70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i376pi = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] pii376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i376i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i99i70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i70i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i8099i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8099i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i99i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ohi7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7075in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] ini7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60, 40,
40, 50, 50, 50, 60, 60, 60 };
public int[] i7075oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i75i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7075i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7075i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7590i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7075i6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6590i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,40, 50, 50, 50, 60, 60, 60, 60,
40, 40, 50, 50, 50, 60, 60, 60 };
}

// Represents a vertex in the graph.
class Vertex {
	public String name; // Vertex name
	public List adj; // Adjacent vertices
	public float time;
	public float nondr;
	public Vertex prev;
	public int scratch;

	float cost;
	float tdaily = 11;
	float tduty = 14;
	float tweekly = 60;
	float trest = 34;
```

```
        float tbreak = 10;

        public double[] bo = new double[] { 8, 17, 2 };
        public double[] ch = new double[] { 9, 17, 3 };
        public double[] in = new double[] { 14, 22, 3 };
        public double[] oh = new double[] { 1, 24, 0 };
        public double[] ny = new double[] { 9, 17, 4 };
        public double[] pa = new double[] { 9, 17, 3 };
        public double[] wa = new double[] { 9, 17, 3 };
        public double[] cha = new double[] { 9, 17, 2 };
        public double[] pi = new double[] { 9, 17, 2 };
        public double[] va = new double[] { 9, 17, 2 };
        float lweekly = 0;
        float ldaily = 0;
        float lduty = 0;

        public Vertex(String nm) {
                name = nm;
                adj = new LinkedList();
                reset();
                this.time = time;
        }

        public Vertex() {

        }

        public void reset() {
                time = Graph.INFINITY;
                prev = null;
                scratch = 0;
        }

}

public class Graph {
        public static final float INFINITY = (float) Double.MAX_VALUE;
        private Map vertexMap = new HashMap(); // Maps String to Vertex

        static HashMap nodeMap = null;


        public void addEdge(String sourceName, String destName, float time) {
                Vertex v = getVertex(sourceName);
                Vertex w = getVertex(destName);
```

```
        v.adj.add(new Edge(w, (time)));
}


private Vertex getVertex(String vertexName) {
        Vertex v = (Vertex) getVertexMap().get(vertexName);
        if (v == null) {
                v = new Vertex(vertexName);
                getVertexMap().put(vertexName, v);
        }
        return v;
}



private void clearAll() {
        for (Iterator itr = getVertexMap().values().iterator(); itr.hasNext();)
        ((Vertex) itr.next()).reset();
}


public float calcTime(Vertex v, Edge e, double dist) {
        double tact = 0.00;
        double tarr = 0.00;
        double tdiff = 0;
        double travelTime = 0;
        int interval = 0;
        tact = v.time % 24;
        String name = (v.name.concat(e.dest.name));
        int t[] = timeDep(name, e);
        tarr = tact + (dist / t[(int) Math.floor(tact)]);
        while (tact > interval) {
                interval++;
        }
        e.ldaily = 0;
        e.lduty = 0;
        e.lweekly = 0;
        e.ldaily = (float) (e.ldaily + v.ldaily);
        e.lduty = (float) (e.lduty + v.lduty);
        e.lweekly = (float) (e.lweekly + v.lweekly);
        if (tarr < interval) {

                travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
                e.ldaily = (float) (e.ldaily + (dist / t[(int) (Math.floor(tact))]));
                e.lduty = (float) (e.lduty + (dist / t[(int) (Math.floor(tact))]));
                e.lweekly = (float) (e.lweekly + (dist / t[(int) (Math.floor(tact))]));
```

```
}

while (tarr > interval) {
        if (interval == 0)
                tdiff = 1;
        else
                tdiff = interval - tact;
        dist = (float) (dist - t[(int) (Math.floor(tact))] * (tdiff));
        travelTime = travelTime + tdiff;
        tact = interval;

        if (e.ldaily + tdiff > e.tdaily) {
                dist = (float) (dist + t[(int) (Math.floor(tact))] * (tdiff));
                dist = dist - t[(int) (Math.floor(tact))]
                                * (e.tdaily - e.ldaily);
                tact = (tact + (e.tdaily - e.ldaily) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tdaily - e.ldaily)- tdiff;
                interval = (int) ((interval + e.tbreak) % 24);
                interval++;
                e.lweekly = e.lweekly + (e.tdaily - e.ldaily);
                e.ldaily = 0;
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;

        } else if (e.lduty + tdiff > e.tduty) {
                dist = (float) (dist + t[(int) (Math.floor(tact))] * (tdiff));
                dist = dist - t[(int) (Math.floor(tact))] * (e.tduty - e.lduty);
                tact = (tact + (e.tduty - e.lduty) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tduty - e.lduty)- tdiff;
                interval = (int) ((interval + e.tbreak) % 24);
                interval++;
                e.ldaily = 0;
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;
        }

        e.ldaily = (float) (e.ldaily + tdiff);
        e.lduty = (float) (e.lduty + tdiff);
        e.lweekly = (float) (e.lweekly + tdiff);

        if (e.ldaily + 1 > e.tdaily) {
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
```

```
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime - (dist / t[(int) (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tdaily - e.ldaily);
                tact = (tact + (e.tdaily - e.ldaily) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tdaily - e.ldaily);
                interval = (int)((interval+e.tbreak)%24);
                interval++;
                e.lweekly = e.lweekly + (e.tdaily -e.ldaily);
                e.ldaily = 0;
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;
            }
        } else if (e.lduty + 1 > e.tduty) {
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime        - (dist / t[(int)
                (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tduty - e.lduty);
                tact = (tact + (e.tduty - e.lduty) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tduty - e.lduty);
                interval =(int)((interval+e.tbreak)% 24);
                interval++;
                e.ldaily = 0;
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;
            }
        } else if (e.lweekly + 1 > e.tweekly) {
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime        - (dist / t[(int)
                (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tweekly - e.lweekly);
                tact = (tact + (e.tweekly - e.lweekly) + e.trest) % 24;
```

```
                        tarr = (tact + (dist / t[(int) (Math.ceil(tact))]));
                        travelTime = travelTime + e.trest + (e.tweekly - e.lweekly);
                        e.lweekly = 0;
                        e.ldaily = 0;
                        e.lduty = 0;
                        interval =(int)((interval+e.trest) % 24);
                        e.nondr = e.nondr + e.trest;

                    }
                } else {
                    tarr = tact + (dist / t[(int) (Math.floor(tact))]);
                    tact = tact % 24;
                    if (tarr < (interval + 1) || tarr == (interval+ 1)) {
                        travelTime = travelTime+ (dist / t[(int) (Math.floor(tact))]);
                        e.ldaily = (float) (e.ldaily + (dist / t[(int) (Math.floor(tact))]));
                        e.lduty = (float) (e.lduty + (dist / t[(int) (Math.floor(tact))]));
                        e.lweekly = (float) (e.lweekly + (dist / t[(int)
                        (Math.floor(tact))]));
                        break;
                    }
                    interval = (interval + 1) % 24;
                }
            }

            return (float) (travelTime);

        }

        @SuppressWarnings("null")
        public int[] timeDep(String f, Edge e) {

            HashMap<String, int[]> tdvrp = new HashMap<String, int[]>();

            int[] updstring = null;
            tdvrp.put("boi84", e.boi84);
            tdvrp.put("i84bo", e.i84bo);
            tdvrp.put("i84i91", e.i84i91);
            tdvrp.put("i91i84", e.i91i84);
            tdvrp.put("i8081i91", e.i8081i91);
            tdvrp.put("i91i8081", e.i91i8081);
            tdvrp.put("i91i9195", e.i91i9195);
            tdvrp.put("i9195i91", e.i9195i91);
            tdvrp.put("i9195ny", e.i9195ny);
            tdvrp.put("nyi9195", e.nyi9195);
            tdvrp.put("boi95", e.boi95);
            tdvrp.put("i95bo", e.i95bo);
```

```
tdvrp.put("i95i9195", e.i95i9195);
tdvrp.put("i9195i95", e.i9195i95);
tdvrp.put("boi87", e.boi87);
tdvrp.put("i87bo", e.i87bo);
tdvrp.put("i87ny", e.i87ny);
tdvrp.put("nyi87", e.nyi87);
tdvrp.put("i87i8690", e.i87i8690);
tdvrp.put("i8690i87", e.i8690i87);
tdvrp.put("i8690i8090", e.i8690i8090);
tdvrp.put("i8090i8690", e.i8090i8690);
tdvrp.put("i87i88", e.i87i88);
tdvrp.put("i88i87", e.i88i87);
tdvrp.put("i88i86", e.i88i86);
tdvrp.put("i86i88", e.i86i88);
tdvrp.put("i86i8690", e.i86i8690);
tdvrp.put("i8690i86", e.i8690i86);
tdvrp.put("nyi8081", e.nyi8081);
tdvrp.put("i8081ny", e.i8081ny);
tdvrp.put("i8081i8099", e.i8081i8099);
tdvrp.put("i8099i8081", e.i8099i8081);
tdvrp.put("i7680i8090", e.i7680i8090);
tdvrp.put("i8090i7680", e.i8090i7680);
tdvrp.put("i76oh", e.i76oh);
tdvrp.put("ohi76", e.ohi76);
tdvrp.put("i7680i76", e.i7680i76);
tdvrp.put("i76i7680", e.i76i7680);
tdvrp.put("i8090i7590", e.i8090i7590);
tdvrp.put("i7590i8090", e.i7590i8090);
tdvrp.put("i78i81", e.i78i81);
tdvrp.put("i81i78", e.i81i78);
tdvrp.put("i81i8176", e.i81i8176);
tdvrp.put("i8176i81", e.i8176i81);
tdvrp.put("i8176i99", e.i8176i99);
tdvrp.put("i99i8176", e.i99i8176);
tdvrp.put("i70oh", e.i70oh);
tdvrp.put("ohi70", e.ohi70);
tdvrp.put("ohin", e.ohin);
tdvrp.put("inoh", e.inoh);
tdvrp.put("ini6590", e.ini6590);
tdvrp.put("i6590in", e.i6590in);
tdvrp.put("i6590ch", e.i6590ch);
tdvrp.put("chi6590", e.chi6590);
tdvrp.put("ini57", e.ini57);
tdvrp.put("i57in", e.i57in);
tdvrp.put("i57i94", e.i57i94);
tdvrp.put("i94i57", e.i94i57);
```

```
tdvrp.put("i94ch", e.i94ch);
tdvrp.put("chi94", e.chi94);
tdvrp.put("ini6970", e.ini6970);
tdvrp.put("i6970in", e.i6970in);
tdvrp.put("i6970i6990", e.i6970i6990);
tdvrp.put("i6990i6970", e.i6990i6970);
tdvrp.put("i6990i6590", e.i6990i6590);
tdvrp.put("i6590i6990", e.i6590i6990);
tdvrp.put("i7590i6990", e.i7590i6990);
tdvrp.put("i6990i7590", e.i6990i7590);
tdvrp.put("ini275", e.ini275);
tdvrp.put("i275in", e.i275in);
tdvrp.put("i275i75", e.i275i75);
tdvrp.put("i75i275", e.i75i275);
tdvrp.put("i75oh", e.i75oh);
tdvrp.put("ohi75", e.ohi75);
tdvrp.put("i75i64", e.i75i64);
tdvrp.put("i64i75", e.i64i75);
tdvrp.put("i64i77", e.i64i77);
tdvrp.put("i77i64", e.i77i64);
tdvrp.put("i77i8177", e.i77i8177);
tdvrp.put("i8177i77", e.i8177i77);
tdvrp.put("i64i40", e.i64i40);
tdvrp.put("i40i64", e.i40i64);
tdvrp.put("i40i81", e.i40i81);
tdvrp.put("i81i40", e.i81i40);
tdvrp.put("i81i8177", e.i81i8177);
tdvrp.put("i8177i81", e.i8177i81);
tdvrp.put("i8177cha", e.i8177cha);
tdvrp.put("chai8177", e.chai8177);
tdvrp.put("i81us74", e.i81us74);
tdvrp.put("us74i81", e.us74i81);
tdvrp.put("us74i85", e.us74i85);
tdvrp.put("i85us74", e.i85us74);
tdvrp.put("i85cha", e.i85cha);
tdvrp.put("chai85", e.chai85);
tdvrp.put("i75i7590", e.i75i7590);
tdvrp.put("i7590i75", e.i7590i75);
tdvrp.put("chachi95", e.chachi95);
tdvrp.put("chi95cha", e.chi95cha);
tdvrp.put("chi95va", e.chi95va);
tdvrp.put("vachi95", e.vachi95);
tdvrp.put("i8177i8164", e.i8177i8164);
tdvrp.put("i8164i8177", e.i8164i8177);
tdvrp.put("i8164va", e.i8164va);
tdvrp.put("vai8164", e.vai8164);
```

```
tdvrp.put("vawa", e.vawa);
tdvrp.put("wava", e.wava);
tdvrp.put("i8164i8166", e.i8164i8166);
tdvrp.put("i8166i8164", e.i8166i8164);
tdvrp.put("i8166wa", e.i8166wa);
tdvrp.put("wai8166", e.wai8166);
tdvrp.put("wawai95", e.wawai95);
tdvrp.put("wai95wa", e.wai95wa);
tdvrp.put("wai95pa", e.wai95pa);
tdvrp.put("pawai95", e.pawai95);
tdvrp.put("i8166i8176", e.i8166i8176);
tdvrp.put("i8176i8166", e.i8176i8166);
tdvrp.put("i8176pa", e.i8176pa);
tdvrp.put("pai8176", e.pai8176);
tdvrp.put("panjt", e.panjt);
tdvrp.put("njtpa", e.njtpa);
tdvrp.put("njtny", e.njtny);
tdvrp.put("nynjt", e.nynjt);
tdvrp.put("i77i8690", e.i77i8690);
tdvrp.put("i8690i77", e.i8690i77);
tdvrp.put("i77i8090", e.i77i8090);
tdvrp.put("i8090i77", e.i8090i77);
tdvrp.put("i81i8081", e.i81i8081);
tdvrp.put("i8081i81", e.i8081i81);
tdvrp.put("i70i376", e.i70i376);
tdvrp.put("i376i70", e.i376i70);
tdvrp.put("i376pi", e.i376pi);
tdvrp.put("pii376", e.pii376);
tdvrp.put("i376i7680", e.i376i7680);
tdvrp.put("i7680i376", e.i7680i376);
tdvrp.put("i99i70", e.i99i70);
tdvrp.put("i70i99", e.i70i99);
tdvrp.put("i8099i7680", e.i8099i7680);
tdvrp.put("i7680i8099", e.i7680i8099);
tdvrp.put("i99i8099", e.i99i8099);
tdvrp.put("i8099i99", e.i8099i99);
tdvrp.put("ohi7075", e.ohi7075);
tdvrp.put("i7075oh", e.i7075oh);
tdvrp.put("i7075in", e.i7075in);
tdvrp.put("ini7075", e.ini7075);
tdvrp.put("i75i7075", e.i75i7075);
tdvrp.put("i7075i75", e.i7075i75);
tdvrp.put("i7075i7590", e.i7075i7590);
tdvrp.put("i7590i7075", e.i7590i7075);
tdvrp.put("i7075i6590", e.i7075i6590);
tdvrp.put("i6590i7075", e.i6590i7075);
```

```
        Set<String> set = tdvrp.keySet();
        Object nodetime;
        Iterator<String> itr = set.iterator();

        while (itr.hasNext()) {
                nodetime = itr.next();
                if (nodetime.equals(f)) {
                        updstring = (int[]) tdvrp.get(nodetime);
                        break;
                }

        }

        return updstring;

}

private final LinkedList settledNodes = new LinkedList();
private final LinkedList unsettledNodes = new LinkedList();
static ArrayList<NodeData> nDataArra = new ArrayList<NodeData>();
int i = 0;
static float tTime = 0;
static ArrayList tr = new ArrayList();

private boolean isSettled(Vertex v) {
        return settledNodes.contains(v);
}

Vertex extractMin() {
        Vertex node;
        Vertex firstelement = (Vertex) unsettledNodes.getFirst();
        Iterator itr = unsettledNodes.iterator();
        while (itr.hasNext()) {
                node = (Vertex) itr.next();
                if (node.time < firstelement.time) {
                        Vertex temp = node;
                        node = firstelement;
                        firstelement = temp;
                }
        }
        firstelement = (Vertex) unsettledNodes.poll();
        return firstelement;
}

public float serveTime(Vertex w) {
        double e;
```

```
double l;
double sn;
double[] interval = null;
double waitingTime;
float rec = 0;
double arr = (w.time % 24);

float tTime = 0;
String f = w.name;
HashMap<String, double[]> service = new HashMap<String, double[]>();
service.put("bo", w.bo);
service.put("ny", w.ny);
service.put("ch", w.ch);
service.put("in", w.in);
service.put("oh", w.oh);
service.put("wa", w.wa);
service.put("pa", w.pa);
service.put("cha", w.cha);
service.put("pi", w.pi);
service.put("va", w.va);

Set<String> set = service.keySet();
Object nodetime;
Iterator<String> itr = set.iterator();

while (itr.hasNext()) {
	nodetime = itr.next();
	if (nodetime.equals(f)) {
		interval = (double[]) service.get(nodetime);
		break;
	}
}
e = interval[0];
l = interval[1];
sn = interval[2];
if (arr < e) {
	waitingTime = e - arr;
	if (w.lduty + waitingTime + sn <= w.tduty) {
		w.lduty = (float) (w.lduty + waitingTime + sn);
		arr = (arr + waitingTime + sn) % 24;
		tTime = (float) (tTime + waitingTime + sn);
		w.nondr = (float) (w.nondr + waitingTime + sn);

		} else if (arr + w.tbreak <= l & arr + w.tbreak > e) {
		w.lduty = 0;
		w.ldaily = 0;
```

```
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);

                } else if (w.lduty + waitingTime + sn > w.tduty) {
                sn = w.tduty - w.lduty;
                w.lduty = 0;
                w.ldaily = 0;
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);
                }

        } else if (arr >= e & arr < l) {
                if (w.lduty + sn >= w.tduty) {
                sn = w.tduty - w.lduty;
                w.lduty = 0;
                w.ldaily = 0;
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);

                } else {
                w.lduty = (float) (w.lduty + sn);
                arr = (arr + sn) % 24;
                tTime = (float) (tTime + sn);
                w.nondr = (float) (w.nondr + sn);

                }
        } else if (arr > l) {
            arr = (arr + w.tbreak) % 24;
            tTime = tTime + w.tbreak;
            w.nondr = (float) (w.nondr + w.tbreak);
            w.lduty = 0;
            w.ldaily = 0;
            if (arr < e) {
                waitingTime = e - arr;
                if (w.lduty + waitingTime + sn <= w.tduty) {
                    w.lduty = (float) (w.lduty + waitingTime + sn);
                    arr = (arr + waitingTime + sn) % 24;
                    Time = (float) (tTime + waitingTime + sn);
                    w.nondr = (float) (w.nondr + waitingTime + sn);

                    } else if (arr + w.tbreak <= l & arr + w.tbreak > e) {
                    w.lduty = 0;
                    w.ldaily = 0;
```

```
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
                        w.nondr = (float) (w.nondr + w.tbreak + sn);

                        } else if (w.lduty + waitingTime + sn > w.tduty) {
                        sn = w.tduty - w.lduty;
                        w.lduty = 0;
                        w.ldaily = 0;
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
                        w.nondr = (float) (w.nondr + w.tbreak + sn);
                        }
                } else if (arr >= e & arr < l) {
                        if (w.lduty + sn >= w.tduty) {
                        sn = w.tduty - w.lduty;
                        w.lduty = 0;
                        w.ldaily = 0;
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
                        w.nondr = (float) (w.nondr + w.tbreak + sn);
                        } else {
                        w.lduty = (float) (w.lduty + sn);
                        arr = (arr + sn) % 24;
                        tTime = (float) (tTime + sn);
                        w.nondr = (float) (w.nondr + sn);
                        }
                }
        }
        return (float) tTime;
}

@SuppressWarnings("null")
ArrayList rec = new ArrayList();

public Vertex dijkstra(Vertex st, String startName, String destName) {

        Vertex start = (Vertex) getVertexMap().get(startName);

        Vertex w = null;
        if (start == null)
        throw new NoSuchElementException("Start vertex not found");
        settledNodes.clear();
        unsettledNodes.clear();
        nDataArray.clear();
        unsettledNodes.add(start);
        if (start.name.equals("oh"))
```

```
        start.time = 7;
else {
        start.time = st.time;
        start.ldaily = st.ldaily;
        start.lduty = st.lduty;
        start.lweekly = st.lweekly;
}

while (!unsettledNodes.isEmpty()) {

        Vertex v = extractMin();

        if (v.name.equals(destName)) {
                if (rec.contains(v.name))
                        break;
                v.time = v.time + serveTime(v);
                st.name = v.name;
                st.time = v.time;
                st.ldaily = v.ldaily;
                st.lduty = v.lduty;
                st.lweekly = v.lweekly;
                rec.add(v.name);

        }

        settledNodes.add(v);
        Iterator itr = v.adj.iterator();
        while (itr.hasNext()) {
                Edge e = (Edge) itr.next();
                w = e.dest;
                double cvw = e.time;

                if (isSettled(w))
                        continue;
                float totalTime = calcTime(v, e, cvw);
                if (w.time > v.time + totalTime) {
                        w.time = (float) (v.time + totalTime);
                        w.ldaily = e.ldaily;
                        w.lduty = e.lduty;
                        w.lweekly = e.lweekly;
                        w.nondr = e.nondr;
                        w.prev = v;
                        unsettledNodes.add(w);
                }

        }
```

```
        }
        return st;
}

public static void main(String[] args) throws IOException {
Vertex source = new Vertex();
Vertex dest = new Vertex();

ArrayList tr = new ArrayList();
String path = "C:\\Users\\viva\\workspace\\TDVRP\\src\\network.txt";
int[] indices;
String[] elements = { "ny", "bo", "pi", "wa", "cha", "va", "ch", "in","pa" };
PermutationGenerator x = new PermutationGenerator(elements.length);
        while (x.hasMore()) {

                String temp[] = new String[elements.length];
                indices = x.getNext();
                for (int i = 0; i < indices.length; i++) {
                        temp[i] = elements[indices[i]];
                }
                String nodes[] = new String[temp.length + 2];

                nodes[0] = "oh";
                for (int j = 0; j < temp.length; j++) {
                        nodes[j + 1] = temp[j];
                }

                nodes[nodes.length - 1] = "oh";

                source.name = nodes[0];
                tr.add(source.name);
                for (int j = 1; j < nodes.length; j++) {
                        // dest.name = nodes[i][j];
                        Graph s = new Graph();
                        FileReader fin = new FileReader(path);
                        BufferedReader graphFile = new BufferedReader(fin);

                        String line;

                        while ((line = graphFile.readLine()) != null) {

                                StringTokenizer st = new StringTokenizer(line);
                                try {
                                        if (st.countTokens() != 3) {
```

```
                                        System.err.println("Skipping ill-formatted line
                                "+ line);
                                        continue;
                                }
                                String source1 = st.nextToken();
                                String dest1 = st.nextToken();
                                int cost = Integer.parseInt(st.nextToken());
                                s.addEdge(source1, dest1, cost);

                        } catch (NumberFormatException e) {
                                System.err.println("Skipping ill-formatted line "+ line);
                        }
                }

                dest = s.dijkstra(source, source.name, nodes[j]);
                source.name = dest.name;
                source.time = dest.time;
                source.ldaily = dest.ldaily;
                source.lduty = dest.lduty;
                source.lweekly = dest.lweekly;
                tr.add(source.name);
        }

        float btime = 164;

        if (source.time < btime) {
                System.out.print("Total Time is :- " + source.time);
                Iterator itr = tr.iterator();
                while (itr.hasNext()) {
                        System.out.print("-" + itr.next());
                }

                System.out.println();
        }
        tr.clear();
    }

}

public void setVertexMap(Map vertexMap) {
    this.vertexMap = vertexMap;
}

public Map getVertexMap() {
    return vertexMap;
}
}
```

APPENDIX B

Simulated Annealing Code:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Deque;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.Random;
import java.util.Set;
import java.util.StringTokenizer;
import java.lang.Math;
import java.lang.reflect.Array;

// Represents an edge in the graph.
class Edge {
	public Vertex dest;
	public Vertex source;
	public float time;
	float nondr;
	float cost;
	float tdaily = 11;
	float tduty = 14;
	float tweekly = 60;
	float trest = 34;
	float tbreak = 10;
	float lweekly = 0;
	float ldaily = 0;
	float lduty = 0;

	public Edge(Vertex d, float c) {
		dest = d;
		time = c;
	}
```

```
public Edge(Vertex s, Vertex d) {
        source = s;
        dest = d;

}

public int[] boi84 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i84bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i84i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i91i84 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i91i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i91i9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195i91 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nyi9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] boi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i95bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i95i9195 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i9195i95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] boi87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i87bo = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i87ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nyi87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i87i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i8690i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i87i88 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i88i87 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i88i86 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i86i88 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i86i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i86 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nyi8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081ny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8099i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i76oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ohi76 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i76 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i76i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7590i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i78i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i78 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i8176i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i99i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i70oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ohi70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6970in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini6970 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ohin = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,
                50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] inoh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,
                50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6590in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6590ch = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini57 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i57in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i57i94 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i94i57 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i94ch = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi94 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i6970 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6970i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6590i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] i7590i6990 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i6990i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ini275 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i275in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i275i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i75i275 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i75oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] ohi75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i75i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i64i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i64i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i77i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i64i40 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i40i64 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i40i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i40 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chai8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81us74 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] us74i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] us74i85 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i85us74 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i85cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chai85 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i75i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7590i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chachi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi95cha = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] chi95va = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vachi95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8177i8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8164i8177 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8164va = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vai8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] vawa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,
                50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wava = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40, 50,
                50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8164i8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166i8164 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166wa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wai8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wawai95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] wai95wa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
public int[] wai95pa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] pawai95 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8166i8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176i8166 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8176pa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] pai8176 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] panjt = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] njtpa = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] njtny = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] nynjt = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8690 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8690i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i77i8090 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8090i77 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i8081i81 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i81i8081 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i70i376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i376i70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i376pi = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] pii376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i376i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i7680i376 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
public int[] i99i70 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
```

```
        public int[] i70i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i8099i7680 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7680i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i8099i99 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i99i8099 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] ohi7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7075in = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] ini7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7075oh = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i75i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7075i75 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30, 40,
                        50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7075i7590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7590i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i7075i6590 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
        public int[] i6590i7075 = new int[] { 60, 60, 60, 60, 60, 60, 50, 30, 30,
                        40, 50, 50, 50, 60, 60, 60, 60, 40, 40, 50, 50, 50, 60, 60, 60 };
}

class Result {
        Vertex node1;

        private Map vertexMap = new HashMap();
        final LinkedList settled = new LinkedList();
        private final LinkedList unsettled = new LinkedList();
        public List adjac = new LinkedList();

        public Result() {

        }

        public void addEdge(Vertex v, Vertex w) {
```

```
        adjac.add(new Edge(v, w));
}

private Vertex getVertex(String vertexName) {
        Vertex v = (Vertex) getVertexMap().get(vertexName);
        if (v == null) {
                v = new Vertex(vertexName);
                getVertexMap().put(vertexName, v);
        }
        return v;
}

public void setVertexMap(Map vertexMap) {
        this.vertexMap = vertexMap;
}

private void clearAll() {
        for (Iterator itr = getVertexMap().values().iterator(); itr.hasNext();)
                ((Vertex) itr.next()).reset();
}

public Map getVertexMap() {
        return vertexMap;
}

public Vertex heuristics(String startName, float a) {

        Vertex v = (Vertex) getVertexMap().get(startName);
        Vertex s = new Vertex();
        Vertex w = new Vertex();
        unsettled.clear();

        Iterator itr = adjac.iterator();
        while (itr.hasNext()) {
                Edge e = (Edge) itr.next();
                s = e.source;
                if (s.name.equals(v.name)) {
                        w = e.dest;
                        w.cost = a * (e.dest.time - e.dest.nondr) + (1 - a)
                                        * e.dest.nondr;

                        if (isSettled(w)) {
                                continue;
                        }
                        unsettled.add(w);
                }
```

```
		}
		v = extractMin();
		settled.add(v);
		return v;
	}

	private boolean isSettled(Vertex v) {
		return settled.contains(v);
	}

	Vertex extractMin() {
		Vertex node;
		Vertex firstelement = (Vertex) unsettled.getFirst();
		Iterator itr = unsettled.iterator();
		while (itr.hasNext()) {
			node = (Vertex) itr.next();
			if (node.cost < firstelement.cost) {
				Vertex temp = node;
				node = firstelement;
				firstelement = temp;

			}

		}

		return firstelement;
	}
}

// Represents a vertex in the graph.
class Vertex {
	public String name; // Vertex name
	public List adj; // Adjacent vertices
	public double time;
	public double nondr;
	public Vertex prev;
	public int scratch;

	double cost;
	float tdaily = 11;
	float tduty = 14;
	float tweekly = 60;
	float trest = 34;
	float tbreak = 10;

	public double[] bo = new double[] { 8, 17, 2 };
```

```
        public double[] ch = new double[] { 9, 17, 3 };
        public double[] in = new double[] { 14, 22, 3 };
        public double[] oh = new double[] { 1, 24, 0 };
        public double[] ny = new double[] { 9, 17, 4 };
        public double[] pa = new double[] { 9, 17, 3 };
        public double[] wa = new double[] { 9, 17, 3 };
        public double[] cha = new double[] { 9, 17, 2 };
        public double[] pi = new double[] { 9, 17, 2 };
        public double[] va = new double[] { 9, 17, 2 };
        double lweekly = 0;
        double ldaily = 0;
        double lduty = 0;

        public Vertex(String nm) {
                name = nm;
                adj = new LinkedList();
                reset();
                this.time = time;
        }

        public Vertex() {

        }

        public void reset() {
                time = Graph.INFINITY;
                prev = null;
                scratch = 0;
        }

}

public class Graph {
        public static final float INFINITY = (float) Double.MAX_VALUE;
        private Map vertexMap = new HashMap(); // Maps String to Vertex
        static Result r = new Result();
        static HashMap nodeMap = null;


        public void addEdge(String sourceName, String destName, float time) {
                Vertex v = getVertex(sourceName);
                Vertex w = getVertex(destName);

                v.adj.add(new Edge(w, (time)));
        }
```

```
@SuppressWarnings("null")
public void printPath(ArrayList nNodeArray, String startName) {
	Vertex v = (Vertex) getVertexMap().get(startName);

	for (int i = 0; i < nNodeArray.size(); i++) {
		NodeData nodeName = (NodeData) nNodeArray.get(i);
		Vertex w = new Vertex();
		w.name = nodeName.getNodeName();
		w.time = nodeName.getNodeTime();
		w.ldaily = nodeName.getLda();
		w.lduty = nodeName.getLdu();
		w.lweekly = nodeName.getLwe();
		w.nondr = nodeName.getNondr();

		r.getVertexMap().put(v.name, v);
		r.getVertexMap().put(w.name, w);
		r.addEdge(v, w);

	}
}


private Vertex getVertex(String vertexName) {
	Vertex v = (Vertex) getVertexMap().get(vertexName);
	if (v == null) {
		v = new Vertex(vertexName);
		getVertexMap().put(vertexName, v);
	}
	return v;
}


private void printPath(Vertex dest) {
	{
		if (dest.prev != null) {
			printPath(dest.prev);

			System.out.print(" to ");
		}
		System.out.print(dest.name);
	}
}


private void clearAll() {
	for (Iterator itr = getVertexMap().values().iterator(); itr.hasNext();)
```

```
                ((Vertex) itr.next()).reset();
}


public float calcTime(Vertex v, Edge e, double dist) {
        double tact = 0.00;
        double tarr = 0.00;
        double tdiff = 0;
        double travelTime = 0;
        int interval = 0;
        tact = v.time % 24;
        String name = (v.name.concat(e.dest.name));
        int t[] = timeDep(name, e);
        tarr = tact + (dist / t[(int) Math.floor(tact)]);
        while (tact > interval) {
                interval++;
        }
        e.ldaily = 0;
        e.lduty = 0;
        e.lweekly = 0;
        e.ldaily = (float) (e.ldaily + v.ldaily);
        e.lduty = (float) (e.lduty + v.lduty);
        e.lweekly = (float) (e.lweekly + v.lweekly);
        if (tarr < interval) {

                travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
                e.ldaily = (float) (e.ldaily + (dist / t[(int) (Math.floor(tact))]));
                e.lduty = (float) (e.lduty + (dist / t[(int) (Math.floor(tact))]));
                e.lweekly = (float) (e.lweekly + (dist / t[(int) (Math.floor(tact))]));

        }

        while (tarr > interval) {
                if (interval == 0)
                        tdiff = 1;
                else
                        tdiff = interval - tact;
                dist = (float) (dist - t[(int) (Math.floor(tact))] * (tdiff));
                travelTime = travelTime + tdiff;
                tact = interval;

                if (e.ldaily + tdiff > e.tdaily) {
                        dist = (float) (dist + t[(int) (Math.floor(tact))] * (tdiff));
                        dist = dist - t[(int) (Math.floor(tact))]
                                        * (e.tdaily - e.ldaily);
```

```
            tact = (tact + (e.tdaily - e.ldaily) + e.tbreak) % 24;
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + e.tbreak + (e.tdaily - e.ldaily)
                        - tdiff;
            interval = (int) ((interval + e.tbreak) % 24);
            interval++;
            e.lweekly = e.lweekly + (e.tdaily - e.ldaily);
            e.ldaily = 0;
            e.lduty = 0;
            e.nondr = e.nondr + e.tbreak;

        } else if (e.lduty + tdiff > e.tduty) {
            dist = (float) (dist + t[(int) (Math.floor(tact))] * (tdiff));
            dist = dist - t[(int) (Math.floor(tact))] * (e.tduty - e.lduty);
            tact = (tact + (e.tduty - e.lduty) + e.tbreak) % 24;
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + e.tbreak + (e.tduty - e.lduty)
                        - tdiff;
            interval = (int) ((interval + e.tbreak) % 24);
            interval++;
            e.ldaily = 0;
            e.lduty = 0;
            e.nondr = e.nondr + e.tbreak;
        }

        e.ldaily = (float) (e.ldaily + tdiff);
        e.lduty = (float) (e.lduty + tdiff);
        e.lweekly = (float) (e.lweekly + tdiff);

        if (e.ldaily + 1 > e.tdaily) {
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime
                            - (dist / t[(int) (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tdaily - e.ldaily);
                tact = (tact + (e.tdaily - e.ldaily) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tdaily - e.ldaily);
                interval = (int) ((interval + e.tbreak) % 24);
                interval++;
                e.lweekly = e.lweekly + (e.tdaily - e.ldaily);
                e.ldaily = 0;
```

```
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;
            }
        } else if (e.lduty + 1 > e.tduty) {
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime
                            - (dist / t[(int) (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tduty - e.lduty);
                tact = (tact + (e.tduty - e.lduty) + e.tbreak) % 24;
                tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
                travelTime = travelTime + e.tbreak + (e.tduty - e.lduty);
                interval = (int) ((interval + e.tbreak) % 24);
                interval++;
                e.ldaily = 0;
                e.lduty = 0;
                e.nondr = e.nondr + e.tbreak;
            }
        } else if (e.lweekly + 1 > e.tweekly) {
            tarr = (tact + (dist / t[(int) (Math.floor(tact))]));
            travelTime = travelTime + (dist / t[(int) (Math.floor(tact))]);
            if (tarr < (interval + 1)) {
                break;
            } else {
                travelTime = travelTime
                            - (dist / t[(int) (Math.floor(tact))]);
                dist = dist - t[(int) (Math.floor(tact))]
                            * (e.tweekly - e.lweekly);
                tact = (tact + (e.tweekly - e.lweekly) + e.trest) % 24;
                tarr = (tact + (dist / t[(int) (Math.ceil(tact))]));
                travelTime = travelTime + e.trest + (e.tweekly - e.lweekly);
                e.lweekly = 0;
                e.ldaily = 0;
                e.lduty = 0;
                interval = (int) ((interval + e.trest) % 24);
                e.nondr = e.nondr + e.trest;

            }
        } else {
            tarr = tact + (dist / t[(int) (Math.floor(tact))]);
            tact = tact % 24;
            if (tarr < (interval + 1) || tarr == (interval + 1)) {
```

```
                    travelTime = travelTime
                                + (dist / t[(int) (Math.floor(tact))]);
                    e.ldaily = (float) (e.ldaily + (dist / t[(int) (Math
                                .floor(tact))]));
                    e.lduty = (float) (e.lduty + (dist / t[(int) (Math
                                .floor(tact))]));
                    e.lweekly = (float) (e.lweekly + (dist / t[(int) (Math
                                .floor(tact))]));
                    break;
                }
                interval = (interval + 1) % 24;
            }
        }

        return (float) (travelTime);

    }

    @SuppressWarnings("null")
    public int[] timeDep(String f, Edge e) {

        HashMap<String, int[]> tdvrp = new HashMap<String, int[]>();

        int[] updstring = null;
        tdvrp.put("boi84", e.boi84);
        tdvrp.put("i84bo", e.i84bo);
        tdvrp.put("i84i91", e.i84i91);
        tdvrp.put("i91i84", e.i91i84);
        tdvrp.put("i8081i91", e.i8081i91);
        tdvrp.put("i91i8081", e.i91i8081);
        tdvrp.put("i91i9195", e.i91i9195);
        tdvrp.put("i9195i91", e.i9195i91);
        tdvrp.put("i9195ny", e.i9195ny);
        tdvrp.put("nyi9195", e.nyi9195);
        tdvrp.put("boi95", e.boi95);
        tdvrp.put("i95bo", e.i95bo);
        tdvrp.put("i95i9195", e.i95i9195);
        tdvrp.put("i9195i95", e.i9195i95);
        tdvrp.put("boi87", e.boi87);
        tdvrp.put("i87bo", e.i87bo);
        tdvrp.put("i87ny", e.i87ny);
        tdvrp.put("nyi87", e.nyi87);
        tdvrp.put("i87i8690", e.i87i8690);
        tdvrp.put("i8690i87", e.i8690i87);
        tdvrp.put("i8690i8090", e.i8690i8090);
        tdvrp.put("i8090i8690", e.i8090i8690);
```

```
tdvrp.put("i87i88", e.i87i88);
tdvrp.put("i88i87", e.i88i87);
tdvrp.put("i88i86", e.i88i86);
tdvrp.put("i86i88", e.i86i88);
tdvrp.put("i86i8690", e.i86i8690);
tdvrp.put("i8690i86", e.i8690i86);
tdvrp.put("nyi8081", e.nyi8081);
tdvrp.put("i8081ny", e.i8081ny);
tdvrp.put("i8081i8099", e.i8081i8099);
tdvrp.put("i8099i8081", e.i8099i8081);
tdvrp.put("i7680i8090", e.i7680i8090);
tdvrp.put("i8090i7680", e.i8090i7680);
tdvrp.put("i76oh", e.i76oh);
tdvrp.put("ohi76", e.ohi76);
tdvrp.put("i7680i76", e.i7680i76);
tdvrp.put("i76i7680", e.i76i7680);
tdvrp.put("i8090i7590", e.i8090i7590);
tdvrp.put("i7590i8090", e.i7590i8090);
tdvrp.put("i78i81", e.i78i81);
tdvrp.put("i81i78", e.i81i78);
tdvrp.put("i81i8176", e.i81i8176);
tdvrp.put("i8176i81", e.i8176i81);
tdvrp.put("i8176i99", e.i8176i99);
tdvrp.put("i99i8176", e.i99i8176);
tdvrp.put("i70oh", e.i70oh);
tdvrp.put("ohi70", e.ohi70);
tdvrp.put("ohin", e.ohin);
tdvrp.put("inoh", e.inoh);
tdvrp.put("ini6590", e.ini6590);
tdvrp.put("i6590in", e.i6590in);
tdvrp.put("i6590ch", e.i6590ch);
tdvrp.put("chi6590", e.chi6590);
tdvrp.put("ini57", e.ini57);
tdvrp.put("i57in", e.i57in);
tdvrp.put("i57i94", e.i57i94);
tdvrp.put("i94i57", e.i94i57);
tdvrp.put("i94ch", e.i94ch);
tdvrp.put("chi94", e.chi94);
tdvrp.put("ini6970", e.ini6970);
tdvrp.put("i6970in", e.i6970in);
tdvrp.put("i6970i6990", e.i6970i6990);
tdvrp.put("i6990i6970", e.i6990i6970);
tdvrp.put("i6990i6590", e.i6990i6590);
tdvrp.put("i6590i6990", e.i6590i6990);
tdvrp.put("i7590i6990", e.i7590i6990);
tdvrp.put("i6990i7590", e.i6990i7590);
```

```
tdvrp.put("ini275", e.ini275);
tdvrp.put("i275in", e.i275in);
tdvrp.put("i275i75", e.i275i75);
tdvrp.put("i75i275", e.i75i275);
tdvrp.put("i75oh", e.i75oh);
tdvrp.put("ohi75", e.ohi75);
tdvrp.put("i75i64", e.i75i64);
tdvrp.put("i64i75", e.i64i75);
tdvrp.put("i64i77", e.i64i77);
tdvrp.put("i77i64", e.i77i64);
tdvrp.put("i77i8177", e.i77i8177);
tdvrp.put("i8177i77", e.i8177i77);
tdvrp.put("i64i40", e.i64i40);
tdvrp.put("i40i64", e.i40i64);
tdvrp.put("i40i81", e.i40i81);
tdvrp.put("i81i40", e.i81i40);
tdvrp.put("i81i8177", e.i81i8177);
tdvrp.put("i8177i81", e.i8177i81);
tdvrp.put("i8177cha", e.i8177cha);
tdvrp.put("chai8177", e.chai8177);
tdvrp.put("i81us74", e.i81us74);
tdvrp.put("us74i81", e.us74i81);
tdvrp.put("us74i85", e.us74i85);
tdvrp.put("i85us74", e.i85us74);
tdvrp.put("i85cha", e.i85cha);
tdvrp.put("chai85", e.chai85);
tdvrp.put("i75i7590", e.i75i7590);
tdvrp.put("i7590i75", e.i7590i75);
tdvrp.put("chachi95", e.chachi95);
tdvrp.put("chi95cha", e.chi95cha);
tdvrp.put("chi95va", e.chi95va);
tdvrp.put("vachi95", e.vachi95);
tdvrp.put("i8177i8164", e.i8177i8164);
tdvrp.put("i8164i8177", e.i8164i8177);
tdvrp.put("i8164va", e.i8164va);
tdvrp.put("vai8164", e.vai8164);
tdvrp.put("vawa", e.vawa);
tdvrp.put("wava", e.wava);
tdvrp.put("i8164i8166", e.i8164i8166);
tdvrp.put("i8166i8164", e.i8166i8164);
tdvrp.put("i8166wa", e.i8166wa);
tdvrp.put("wai8166", e.wai8166);
tdvrp.put("wawai95", e.wawai95);
tdvrp.put("wai95wa", e.wai95wa);
tdvrp.put("wai95pa", e.wai95pa);
tdvrp.put("pawai95", e.pawai95);
```

```
tdvrp.put("i8166i8176", e.i8166i8176);
tdvrp.put("i8176i8166", e.i8176i8166);
tdvrp.put("i8176pa", e.i8176pa);
tdvrp.put("pai8176", e.pai8176);
tdvrp.put("panjt", e.panjt);
tdvrp.put("njtpa", e.njtpa);
tdvrp.put("njtny", e.njtny);
tdvrp.put("nynjt", e.nynjt);
tdvrp.put("i77i8690", e.i77i8690);
tdvrp.put("i8690i77", e.i8690i77);
tdvrp.put("i77i8090", e.i77i8090);
tdvrp.put("i8090i77", e.i8090i77);
tdvrp.put("i81i8081", e.i81i8081);
tdvrp.put("i8081i81", e.i8081i81);
tdvrp.put("i70i376", e.i70i376);
tdvrp.put("i376i70", e.i376i70);
tdvrp.put("i376pi", e.i376pi);
tdvrp.put("pii376", e.pii376);
tdvrp.put("i376i7680", e.i376i7680);
tdvrp.put("i7680i376", e.i7680i376);
tdvrp.put("i99i70", e.i99i70);
tdvrp.put("i70i99", e.i70i99);
tdvrp.put("i8099i7680", e.i8099i7680);
tdvrp.put("i7680i8099", e.i7680i8099);
tdvrp.put("i99i8099", e.i99i8099);
tdvrp.put("i8099i99", e.i8099i99);
tdvrp.put("ohi7075", e.ohi7075);
tdvrp.put("i7075oh", e.i7075oh);
tdvrp.put("i7075in", e.i7075in);
tdvrp.put("ini7075", e.ini7075);
tdvrp.put("i75i7075", e.i75i7075);
tdvrp.put("i7075i75", e.i7075i75);
tdvrp.put("i7075i7590", e.i7075i7590);
tdvrp.put("i7590i7075", e.i7590i7075);
tdvrp.put("i7075i6590", e.i7075i6590);
tdvrp.put("i6590i7075", e.i6590i7075);
Set<String> set = tdvrp.keySet();
Object nodetime;
Iterator<String> itr = set.iterator();

while (itr.hasNext()) {
        nodetime = itr.next();
        if (nodetime.equals(f)) {
                updstring = (int[]) tdvrp.get(nodetime);
                break;
        }
```

```
        }

        return updstring;

    }

    private final LinkedList settledNodes = new LinkedList();
    private final LinkedList unsettledNodes = new LinkedList();
    static ArrayList<NodeData> nDataArray = new ArrayList<NodeData>();
    int i = 0;
    static double tTime = 0;
    static ArrayList tr = new ArrayList();

    public void dijkstra2(Vertex st, String startName) {

        Vertex start = (Vertex) getVertexMap().get(startName);
        Vertex w = null;
        ArrayList rec = new ArrayList();
        if (start == null)
            throw new NoSuchElementException("Start vertex not found");
        settledNodes.clear();
        unsettledNodes.clear();
        nDataArray.clear();
        rec.clear();
        unsettledNodes.add(start);
        if (start.name.equals("oh"))
            start.time = 7;
        else {
            start.time = st.time;
            start.ldaily = st.ldaily;
            start.lduty = st.lduty;
            start.lweekly = st.lweekly;
        }

        while (!unsettledNodes.isEmpty()) {

            Vertex v = extractMin();

            String n[] = { "oh", "bo", "ny", "wa", "cha", "pi", "pa", "va",
                           "ch", "in" };
            int j = tr.size();
            if (j == 10) {
                if (v.name.equals(n[0])) {
                    NodeData nData = new NodeData();
                    nData.setNodeName(v.name);
```

```
                nData.setLda(v.ldaily);

                nData.setLwe(v.lweekly);
                v.time = v.time + serveTime(v);
                tTime = v.time;
                nData.setLdu(v.lduty);
                nData.setNodeTime(v.time);
                nData.setNondr(v.nondr);
                nDataArray.add(nData);
                System.out.println("Cost is:" + v.time);
                printPath(v);

                System.out.println();
        }
} else {
        for (int i = 1; i < Array.getLength(n); i++) {
                if (v.name.equals(n[i])) {
                        if (tr.contains(v.name))
                                break;
                        if (rec.contains(v.name))
                                break;
                        NodeData nData = new NodeData();
                        rec.add(v.name);
                        nData.setNodeName(v.name);
                        nData.setLda(v.ldaily);

                        nData.setLwe(v.lweekly);
                        v.time = v.time + serveTime(v);
                        nData.setNodeTime(v.time);
                        nData.setLdu(v.lduty);
                        nData.setNondr(v.nondr);
                        nDataArray.add(nData);
                        System.out.println("Cost is:" + v.time);
                        printPath(v);
                        System.out.println();
                }
        }
}
settledNodes.add(v);
Iterator itr = v.adj.iterator();
while (itr.hasNext()) {
        Edge e = (Edge) itr.next();
        w = e.dest;
        double cvw = e.time;

        if (cvw < 0)
```

```
                        try {
                                throw new Exception("Graph has negative edges");
                        } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                if (isSettled(w))
                        continue;
                float totalTime = calcTime(v, e, cvw);
                if (w.time > v.time + totalTime) {
                        w.time = (float) (v.time + totalTime);
                        w.ldaily = e.ldaily;
                        w.lduty = e.lduty;
                        w.lweekly = e.lweekly;
                        w.nondr = e.nondr;
                        w.prev = v;
                        unsettledNodes.add(w);
                }

        }

    }

}

private boolean isSettled(Vertex v) {
        return settledNodes.contains(v);
}

Vertex extractMin() {
        Vertex node;
        Vertex firstelement = (Vertex) unsettledNodes.getFirst();
        Iterator itr = unsettledNodes.iterator();
        while (itr.hasNext()) {
                node = (Vertex) itr.next();
                if (node.time < firstelement.time) {
                        Vertex temp = node;
                        node = firstelement;
                        firstelement = temp;

                }

        }
        firstelement = (Vertex) unsettledNodes.poll();
        return firstelement;
}
```

```
public float serveTime(Vertex w) {
        double e;
        double l;
        double sn;
        double[] interval = null;
        double waitingTime;
        float rec = 0;
        double arr = (w.time % 24);

        float tTime = 0;
        String f = w.name;
        HashMap<String, double[]> service = new HashMap<String, double[]>();
        service.put("bo", w.bo);
        service.put("ny", w.ny);
        service.put("ch", w.ch);
        service.put("in", w.in);
        service.put("oh", w.oh);
        service.put("wa", w.wa);
        service.put("pa", w.pa);
        service.put("cha", w.cha);
        service.put("pi", w.pi);
        service.put("va", w.va);

        Set<String> set = service.keySet();
        Object nodetime;
        Iterator<String> itr = set.iterator();

        while (itr.hasNext()) {
                nodetime = itr.next();
                if (nodetime.equals(f)) {
                        interval = (double[]) service.get(nodetime);
                        break;
                }
        }
        e = interval[0];
        l = interval[1];
        sn = interval[2];
        if (arr < e) {
                waitingTime = e - arr;
                if (w.lduty + waitingTime + sn <= w.tduty) {
                        w.lduty = (float) (w.lduty + waitingTime + sn);
                        arr = (arr + waitingTime + sn) % 24;
                        tTime = (float) (tTime + waitingTime + sn);
                        w.nondr = (float) (w.nondr + waitingTime + sn);
                } else if (arr + w.tbreak <= l & arr + w.tbreak > e) {
```

```
                w.lduty = 0;
                w.ldaily = 0;
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);
        } else if (w.lduty + waitingTime + sn > w.tduty) {
                sn = w.tduty - w.lduty;
                w.lduty = 0;
                w.ldaily = 0;
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);
        }
} else if (arr >= e & arr < l) {
        if (w.lduty + sn >= w.tduty) {
                sn = w.tduty - w.lduty;
                w.lduty = 0;
                w.ldaily = 0;
                arr = (arr + w.tbreak + sn) % 24;
                tTime = (float) (tTime + w.tbreak + sn);
                w.nondr = (float) (w.nondr + w.tbreak + sn);
        } else {
                w.lduty = (float) (w.lduty + sn);
                arr = (arr + sn) % 24;
                tTime = (float) (tTime + sn);
                w.nondr = (float) (w.nondr + sn);

        }
} else if (arr > l) {
        arr = (arr + w.tbreak) % 24;
        tTime = tTime + w.tbreak;
        w.nondr = (float) (w.nondr + w.tbreak);
        w.lduty = 0;
        w.ldaily = 0;
        if (arr < e) {
                waitingTime = e - arr;
                if (w.lduty + waitingTime + sn <= w.tduty) {
                        w.lduty = (float) (w.lduty + waitingTime + sn);
                        arr = (arr + waitingTime + sn) % 24;
                        tTime = (float) (tTime + waitingTime + sn);
                        w.nondr = (float) (w.nondr + waitingTime + sn);
                } else if (arr + w.tbreak <= l & arr + w.tbreak > e) {
                        w.lduty = 0;
                        w.ldaily = 0;
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
```

```
                        w.nondr = (float) (w.nondr + w.tbreak + sn);
                    } else if (w.lduty + waitingTime + sn > w.tduty) {
                        sn = w.tduty - w.lduty;
                        w.lduty = 0;
                        w.ldaily = 0;
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
                        w.nondr = (float) (w.nondr + w.tbreak + sn);
                    }
                } else if (arr >= e & arr < l) {
                    if (w.lduty + sn >= w.tduty) {
                        sn = w.tduty - w.lduty;
                        w.lduty = 0;
                        w.ldaily = 0;
                        arr = (arr + w.tbreak + sn) % 24;
                        tTime = (float) (tTime + w.tbreak + sn);
                        w.nondr = (float) (w.nondr + w.tbreak + sn);
                    } else {
                        w.lduty = (float) (w.lduty + sn);
                        arr = (arr + sn) % 24;
                        tTime = (float) (tTime + sn);
                        w.nondr = (float) (w.nondr + sn);

                    }
                }
            }
            return (float) tTime;
        }

        @SuppressWarnings("null")
        ArrayList rec = new ArrayList();

        public Vertex dijkstra(Vertex st, String startName, String destName) {

            Vertex start = (Vertex) getVertexMap().get(startName);

            Vertex w = null;
            if (start == null)
                throw new NoSuchElementException("Start vertex not found");
            settledNodes.clear();
            unsettledNodes.clear();
            nDataArray.clear();
            unsettledNodes.add(start);
            if (start.name.equals("oh"))
                start.time = 7;
            else {
```

```
            start.time = st.time;
            start.ldaily = st.ldaily;
            start.lduty = st.lduty;
            start.lweekly = st.lweekly;
        }

        while (!unsettledNodes.isEmpty()) {

            Vertex v = extractMin();

            if (v.name.equals(destName)) {
                if (rec.contains(v.name))
                    break;
                v.time = v.time + serveTime(v);
                st.name = v.name;
                st.time = v.time;
                st.ldaily = v.ldaily;
                st.lduty = v.lduty;
                st.lweekly = v.lweekly;
                rec.add(v.name);

            }

            settledNodes.add(v);
            Iterator itr = v.adj.iterator();
            while (itr.hasNext()) {
                Edge e = (Edge) itr.next();
                w = e.dest;
                double cvw = e.time;

                if (cvw < 0)
                    try {
                        throw new Exception("Graph has negative edges");
                    } catch (Exception e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                    }
                if (isSettled(w))
                    continue;
                float totalTime = calcTime(v, e, cvw);
                if (w.time > v.time + totalTime) {
                    w.time = (float) (v.time + totalTime);
                    w.ldaily = e.ldaily;
                    w.lduty = e.lduty;
                    w.lweekly = e.lweekly;
                    w.nondr = e.nondr;
```

```
                        w.prev = v;
                        unsettledNodes.add(w);
                    }

                }

            }
            return st;
        }

        public static void main(String[] args) throws IOException {
            Graph g = new Graph();
            int i = 0;
            float a;
            String destName = null;
            String startName = null;
            String startNode = "oh";
            Vertex start = null;
            Vertex rec = null;
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Enter start node:");
            startName = in.readLine();
            try {
                String path = "C:\\Users\\viva\\workspace\\TDVRP\\src\\network.txt";

                a = (float) 0.9;
                int j = tr.size();
                while (j != 11) {

                    g = new Graph();
                    FileReader fin = new FileReader(path);
                    BufferedReader graphFile = new BufferedReader(fin);

                    String line;

                    while ((line = graphFile.readLine()) != null) {

                        StringTokenizer st = new StringTokenizer(line);

                        try {
                            if (st.countTokens() != 3) {
                                System.err.println("Skipping ill-formatted line "
                                        + line);
                                continue;
                            }
                            String source = st.nextToken();
```

```
                        String dest = st.nextToken();
                        int cost = Integer.parseInt(st.nextToken());
                        g.addEdge(source, dest, cost);

                } catch (NumberFormatException e) {
                        System.err.println("Skipping ill-formatted line "
                                        + line);
                }
        }
        if (startName.equals(startNode)) {
                if (startName.equals("oh"))
                        start = (Vertex) g.getVertexMap().get(startName);
                else
                        start = rec;

                if (g.vertexMap.containsKey(startName)) {
                        tr.add(start.name);
                        g.dijkstra2(start, startName);
                        g.printPath(nDataArray, startName);
                        rec = r.heuristics(startName, a);
                        startName = rec.name;
                        startNode = rec.name;
                }
        }
        j++;
}
Iterator itr = tr.iterator();
String node;
while (itr.hasNext()) {
        node = (String) itr.next();
        System.out.print(node + " - ");

}
System.out.print(" Total Time is : " + tTime);
double te = 3000;
double bTime = tTime;
Vertex source = new Vertex();
LinkedList br = new LinkedList();
double fTime = 0;
while (te > 0.01) {
        Random generator = new Random();
        int r1 = generator.nextInt(9) + 1;

        int r2 = generator.nextInt(9) + 1;

        if (r1 == r2) {
```

```
            r1 = generator.nextInt(9) + 1;
            r2 = generator.nextInt(9) + 1;
        }
        double r = generator.nextDouble();
        Object ar[] = tr.toArray();
        Object temp = ar[r1];
        ar[r1] = ar[r2];
        ar[r2] = temp;
        LinkedList t = new LinkedList();
        for (int m = 0; m < ar.length; m++) {
            t.add(ar[m]);
        }
        Iterator itr1 = t.iterator();

        source.name = (String) itr1.next();
        while (itr1.hasNext()) {
            Vertex dest = new Vertex();
            dest.name = (String) itr1.next();

            Graph s = new Graph();
            FileReader fin = new FileReader(path);
            BufferedReader graphFile = new BufferedReader(fin);

            String line;

            while ((line = graphFile.readLine()) != null) {

                StringTokenizer st = new StringTokenizer(line);
                try {
                    if (st.countTokens() != 3) {
                        System.err.println("Skipping ill-
                        formatted line "+ line);
                        continue;
                    }
                    String source1 = st.nextToken();
                    String dest1 = st.nextToken();
                    int cost = Integer.parseInt(st.nextToken());
                    s.addEdge(source1, dest1, cost);

                } catch (NumberFormatException e) {
                    System.err.println("Skipping ill-formatted line "
                                + line);
                }
            }

            dest = s.dijkstra(source, source.name, dest.name);
```

```
                source.name = dest.name;
                source.time = dest.time;
                source.ldaily = dest.ldaily;
                source.lduty = dest.lduty;
                source.lweekly = dest.lweekly;
                fTime = dest.time;
        }

        if (fTime < tTime) {
                tr.clear();

                Iterator itr11 = t.iterator();
                while (itr11.hasNext()) {
                        tr.add(itr11.next());
                }
                tTime = fTime;
        } else if (r < Math.exp((tTime - fTime) / te)) {
                tr.clear();
                Iterator itr11 = t.iterator();
                while (itr11.hasNext()) {
                        tr.add(itr11.next());
                }
                tTime = fTime;
        }
        if (tTime < bTime) {
                Iterator itr11 = t.iterator();
                br.clear();
                while (itr11.hasNext()) {
                        br.add(itr11.next());
                        bTime = tTime;
                }
        }
        te = (0.95) * te;
}
System.out.println();
Iterator itr111 = br.iterator();
String node1;
while (itr111.hasNext()) {
        node1 = (String) itr111.next();
        System.out.print(node1 + " - ");

}
System.out.print("Total Time is:" + bTime);

tr.clear();
```

```
		} catch (IOException e) {
			System.err.println(e);
		}

	}

	public void setVertexMap(Map vertexMap) {
		this.vertexMap = vertexMap;
	}

	public Map getVertexMap() {
		return vertexMap;
	}
}
```

APPENDIX C

Network Data:

bo i84 55
i84 bo 55
i84 i91 42
i91 i84 42
i8081 i91 91
i91 i8081 91
i91 i9195 37
i9195 i91 37
i9195 ny 75
ny i9195 75
bo i95 18
i95 bo 18
i95 i9195 60
i9195 i95 60
bo i87 151
i87 bo 151
i87 ny 118
ny i87 118
i87 i8690 371
i8690 i87 371
i8690 i8090 62
i8090 i8690 62
i87 i88 25
i88 i87 25
i88 i86 176
i86 i88 176
i86 i8690 193
i8690 i86 193
ny i8081 113
i8081 ny 113
i8099 i7680 75
i7680 i8099 75
i7680 i8090 100
i8090 i7680 100
i76 oh 100
oh i76 100
i7680 i76 62
i76 i7680 62
i8090 i7590 146
i7590 i8090 146

i78 i81 137
i81 i78 137
i81 i8176 38
i8176 i81 38
i8176 i99 63
i99 i8176 63
i70 oh 190
oh i70 190
i7075 in 107
oh i7075 63
i7075 oh 63
i7075 i7590 119
i7590 i7075 119
in i7075 107
in i6590 148
i6590 in 148
i6590 ch 32
ch i6590 32
in i57 114
i57 in 114
i57 i94 122
i94 i57 122
i94 ch 12
ch i94 12
in i6970 15
i6970 in 15
i6970 i6990 155
i6990 i6970 155
i6990 i6590 124
i6590 i6990 124
i7590 i6990 85
i6990 i7590 85
in i275 92
i275 in 92
i275 i75 24
i75 i275 24
i75 oh 105
oh i75 105
i75 i64 73
i64 i75 73
i64 i77 229
i77 i64 229
i77 i8177 66
i8177 i77 66
i64 i40 172
i40 i64 172

i40 i81 29
i81 i40 29
i81 i8177 155
i8177 i81 155
i8177 cha 135
cha i8177 135
i81 us74 76
us74 i81 76
us74 i85 90
i85 us74 90
i85 cha 27
cha i85 27
i75 i7075 68
i7075 i75 68
cha chi95 264
chi95 cha 264
chi95 va 25
va chi95 25
i8177 i8164 140
i8164 i8177 140
i8164 va 102
va i8164 102
va wa 105
wa va 105
i8164 i8166 79
i8166 i8164 79
i8166 wa 76
wa i8166 76
wa wai95 35
wai95 wa 35
wai95 pa 88
pa wai95 88
i8166 i8176 126
i8176 i8166 126
i8176 pa 102
pa i8176 102
pa njt 12
njt pa 12
njt ny 83
ny njt 83
i77 i8690 340
i8690 i77 340
i77 i8090 235
i8090 i77 235
i81 i8081 99
i8081 i81 99

i70 i376 20
i376 i70 20
i376 pi 15
pi i376 15
i376 i7680 77
i7680 i376 77
i99 i70 89
i70 i99 89
i99 i8099 84
i8099 i99 84
i8099 i8081 197
i8081 i8099 197
i7075 i6590 320
i6590 i7075 320