IMPLEMENTATION OF THE METAHEURISTIC
TABU SEARCH IN ROUTE SELECTION
FOR MOBILITY ANALYSIS SUPPORT SYSTEM

THESIS

David M. Ryer, Major, USAF

AFIT/GOA/ENS/99M-07

Approved for public release; distribution unlimited
THESIS APPROVAL

NAME: David M. Ryer, Major, USAF       CLASS: GOA-99M

THESIS TITLE: Implementation of the Metaheuristic Tabu Search in Route Selection for Mobility Analysis Support System

DEFENSE DATE: 2 March 1999

COMMITTEE:     NAME/TITLE/DEPARTMENT

Advisor        T. Glenn Bailey, Lieutenant Colonel, USAF
                Assistant Professor of Operations Research
                Department of Operational Sciences
                Air Force Institute of Technology

Reader         James T. Moore
                Associate Professor of Operations Research
                Department of Operational Sciences
                Air Force Institute of Technology

Reader         William B. Carlton, Lieutenant Colonel (P), USA
                Adjunct Assistant Professor of Operations Research
                Department of Operational Sciences
                Air Force Institute of Technology

SIGNATURE

[Signatures]

[Signatures]
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.
IMPLEMENTATION OF THE METAHEURISTIC
TABU SEARCH IN ROUTE SELECTION
FOR MOBILITY ANALYSIS SUPPORT SYSTEM

THESIS

Presented to the Faculty of the Graduate School of Engineering
Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

David M. Ryer, B.S.
Major, USAF

March 1999

Approved for public release; distribution unlimited
Acknowledgements

This project and the year and a half of school that accompanied it would not have been possible without the love and support of my wife, Rita. I hope I am able to return the favor to you someday soon. My son, Shaun, is a perfect blessing that helped remind me what is important and how to have fun.

I must also thank my advisor, Lt Col Glenn Bailey, whose direction and ideas always gave me the right focus. I would also like to thank my committee, LTC Carlton for helping me extend his initial work and Dr. Moore for his constructive reviews. Finally, I would like to thank my partner, Capt Kevin O’Rourke whose programming expertise was invaluable in completing this “straightforward” coding project.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>Abstract</td>
<td>vi</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Scope</td>
<td>4</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Air Mobility Command's Mobility Model - MASS</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Problem Formulation – The Vehicle Routing Problem</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Methodology</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1 Tour Structure</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Starting Solution</td>
<td>12</td>
</tr>
<tr>
<td>2.3.3 Solution Neighborhood</td>
<td>13</td>
</tr>
<tr>
<td>2.3.4 Tabu Criteria</td>
<td>15</td>
</tr>
<tr>
<td>2.3.5 Algorithm Complexity</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Testing and Validation</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Extensions for Solving Mobility Routing Problems</td>
<td>24</td>
</tr>
<tr>
<td>2.6 Dimensions of the Mobility Routing Problem</td>
<td>26</td>
</tr>
<tr>
<td>2.7 Future Research</td>
<td>33</td>
</tr>
<tr>
<td>2.8 Conclusion</td>
<td>34</td>
</tr>
<tr>
<td>Appendix A: Extended Problem Formulation</td>
<td>37</td>
</tr>
<tr>
<td>A.1 Traveling Salesman Problem</td>
<td>38</td>
</tr>
<tr>
<td>A.2 Multiple Traveling Salesman Problem</td>
<td>40</td>
</tr>
<tr>
<td>A.3 Vehicle Routing Problem</td>
<td>41</td>
</tr>
<tr>
<td>A.4 Multiple Depot VRP</td>
<td>44</td>
</tr>
<tr>
<td>A.5 Pickup and Delivery Problem</td>
<td>45</td>
</tr>
<tr>
<td>Appendix B: Java Documentation</td>
<td>49</td>
</tr>
<tr>
<td>Bibliography</td>
<td>99</td>
</tr>
<tr>
<td>Vita</td>
<td>102</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. MASS Regions</td>
<td>6</td>
</tr>
<tr>
<td>2. Routes that Link Region Pairs</td>
<td>6</td>
</tr>
<tr>
<td>3. Disjunctive Graph Representation of Tour</td>
<td>12</td>
</tr>
<tr>
<td>4. RTS Pseudocode</td>
<td>16</td>
</tr>
<tr>
<td>5. Calculation of Groundspeed to Account for Winds</td>
<td>25</td>
</tr>
<tr>
<td>6. Solution of Simple TSP comprised of the 50 U. S. Capitals</td>
<td>29</td>
</tr>
<tr>
<td>7. Mobility Problem Constraints and their Effect on Route Selection</td>
<td>30</td>
</tr>
<tr>
<td>8. Southwest Asia Scenario</td>
<td>32</td>
</tr>
<tr>
<td>9. Pacific Scenario</td>
<td>33</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Solomon mTSPTW 25 Customers</td>
<td>18</td>
</tr>
<tr>
<td>2. Solomon mTSPTW 50 Customers</td>
<td>19</td>
</tr>
<tr>
<td>3. Solomon mTSPTW 100 Customers</td>
<td>20</td>
</tr>
<tr>
<td>4. Solomon VRPTW 25 Customers</td>
<td>21</td>
</tr>
<tr>
<td>5. Solomon VRPTW 50 Customers</td>
<td>22</td>
</tr>
<tr>
<td>6. Solomon VRPTW 100 Customers</td>
<td>23</td>
</tr>
</tbody>
</table>
Abstract

This thesis employs a reactive tabu search heuristic implemented in the Java programming language to solve a real world variation of the vehicle routing problem with the objective of providing quality routes to Mobility Analysis Support System (MASS). MASS is a stochastic simulation model used extensively by Air Mobility Command (AMC) to analyze strategic airlift capabilities and future procurement decisions. This dynamic real world problem of strategic and tactical airlift possesses a number of side constraints such as vehicle capacities, route length and time windows in a sizeable network with multiple depots and a large fleet of heterogeneous vehicles. Finding optimal solutions to this problem is currently not practical. Currently, MASS requires all possible routes used in its simulation to be manually selected. As a result, the route selection process is a tedious and time consuming process that relies on experience and past performance of the model to obtain quality routes for the mobility system.
Chapter 1

1.1 Introduction

Mobility Analysis Support System (MASS) is a simulation model used extensively by Air Mobility Command (AMC) to analyze strategic airlift capabilities and future procurement decisions, whose routes are calculated by an experienced analyst through trial and error. This thesis employs a reactive tabu search heuristic implemented in the Java programming language to solve the vehicle routing problem with the objective of providing quality routes to MASS that are as good or better than those currently used.

1.2 Background

Most vehicle routing problems (VRP’s) are NP-hard combinatorial problems for which no polynomially bounded algorithm has yet been found (Baker 1986). Convergent algorithms can rarely solve problems larger than 50 customers, and often require relatively few side constraints (Gendreau et al. 1997). Unfortunately, real world problems such as strategic airlift possess a number of side constraints such as precedence, route and vehicle capacities, route length and time windows in a sizeable network with multiple depots, and a large fleet of heterogeneous vehicles. Therefore, finding optimal solutions using such techniques as branch and bound or dynamic programming is currently not practical.

On the other hand, many heuristic approaches can provide excellent solutions with reasonable computational times. Greedy algorithms, which prove to be very useful in simpler problems, fail to achieve the desired results with respect to solution quality, while simulated annealing (SA) displays large variance with regard to computational time
and quality due to the random nature of its search strategy (Osman 1993). Genetic algorithms (GAs) are difficult to apply to VRP's with capacity, distance, and time window constraints because they were designed to solve numerical optimization problems rather than combinatorial optimization problems (Gendreau et al. 1997). Conversely, tabu search (TS) has provided excellent results on this type of problem with the implementation of intensification and diversification strategies (Gendreau et al. 1997). Intensification uses choice rules to encourage move combinations that incorporate good solution features, while diversification forces the solution search to unexplored regions or to solutions significantly different than those already found (Glover and Laguna 1997). The literature shows TS is a robust approach to solving many variations of the VRP and dominates current studies of routing problems (Gendreau et al. 1997, Xu and Kelly 1996, Rochat and Semet 1994, Renaud et al. 1996, Osman 1993, Garcia et al. 1994, Chiang and Russell 1997, Carlton 1995).

Recent modeling efforts in the military airlift community emphasize simulation over optimization, in part due to the ease in which simulation can represent the stochastic nature of the problems being studied (Rosenthal et al. 1997, Morton et al. 1996). However, more recent efforts look at combining simulation and optimization, particularly with regard to the Air Mobility Command's legacy model Mobility Analysis Support System (MASS). MASS simulates the strategic airlift environment for analysis of doctrine, strategic airlift capability, current AMC airlift assets and future AMC acquisitions. This simulation analysis supports the activities of the AMC Commander, the United States Transportation Command (USTRANSCOM), and a wide variety of theater and campaign level commanders. In addition, proprietary organizations like
Lockheed-Martin and Boeing use MASS to analyze future airlift systems. Possessing a global domain, MASS simulates up to 300 bases at any latitude and longitude in the world, using up to ten types of aircraft, with the entire fleet of strategic airlift aircraft tracked by tail number and cargo classified by weight, dimension and special handling instructions (Boeing 1996). In short, the model's domain is the world and it spans all strategic aircraft in the USAF inventory and CRAF with virtually every cargo combination (Boeing 1996).

The primary component of MASS is the Airlift Flow Model (AFM), which orchestrates the simulation of mission events throughout the entire system. Supporting this core element are various loading, ground crew, command and control, and tanker models. Current validation efforts include output comparisons between MASS and the Naval Postgraduate School/RAND Mobility Optimizer (NRMO) by crossfeeding relative information between the two models in a series of repetitive simulations and then observing if the two models converge on the same solution (Wright 1998).

Extending the work of Ryan (1999) and Carlton (1995), we implement the metaheuristic of reactive TS (RTS) in an object-oriented (OO) programming language. The RTS route solution represents the input to MASS for comparison with current routing selection methods. Our goal is to improve the route selection process used for MASS by using RTS-based routing inputs instead of NRMO or manually derived routing solutions.
1.3 Scope

Earlier attempts at route generators employ the optimal k-shortest path method and route length restrictions representing aircraft type maximum flight legs. This effort, coded in two separate computer-programming languages, has shown limited results in large realistic scenarios (Rink 1998). Extending this effort to include additional route selection criteria requires an efficient and robust method currently not achievable by convergent algorithms. In order to improve the overall quality of route selection, AMC Studies and Analysis (XPY) proposes adding international airspace routing constraints, crew staging and air-refueling constraints to the routing problem formulation.

Following the hierarchy scheme introduced by Carlton (1995) this problem can be treated as a VRP with multiple depots (MD), multiple non-homogenous vehicles (MHV), and route length constraints (RL). As with most heuristic techniques, the algorithm, once constructed, will have to be fine-tuned to accurately represent the most important routing considerations as modeled by MASS.
Chapter 2

2.1 Air Mobility Command’s Mobility Model - MASS

Currently route segments are fed into MASS in an ordered list (one of many input files required for a single simulation run) determined solely by the user. The Aircraft Routing Algorithm (ARA) of MASS checks these route segments in file order for a feasible crew plan. Then the Aircraft Flight Plan Algorithm (AFPA) determines if a route can be feasibly mission planned (flying hour availability, aircraft target use rate, route length, ramp space or maximum on ground (MOG)) (Boeing 1996). If this route segment is not feasible, then the next route in the file is checked. If no feasible route segment is found, the planning phase returns to a previous planned enroute segment and restarts the process. If no feasible crew plan or mission plan exists on the routes provided, the aircraft is scheduled for a “Part IV” mission; i.e., it flies from its present position to its home station base as a recovery.

Because of the importance of crew feasibility in MASS, a constraint to this problem prioritizes routes with available crews to avoid unnecessary recovery missions. Listed in the order of importance, the following considerations must be evaluated by any route generator: distance, route length restriction, crew availability, route or airspace restrictions, winds, and air refueling capability. All locations that make up a route (Home Station, Onload, Offload, Enroute, and Recovery) are further characterized by the geographical region in which they are located (Figure 1).
In order to avoid the task of explicitly listing all possible route permutations from each on-load base to each off-load base, the Airlift Flow Model (AFM) deals with region pairs. With this representation, it is not necessary to specify every possible route joining the departure base to the destination base, but instead only the routes joining the respective regions (Brigantic 1998).
2.2 Problem Formulation – The Vehicle Routing Problem

The VRP can be viewed as an extension of the basic traveling salesman problem (TSP) that adds capacity constraints to multiple salesman or vehicles. (For a more in-depth discussion on building the formulation for this family of problems see Appendix A.) The VRP involves \( w \) vehicles leaving a depot and servicing \( n \) customers, each with a unique demand \( d_i \). Each vehicle \( v \) has a limited capacity \( K_v \) and maximum time length for a route \( T_v \) that constrains their closed delivery routes. This particular instance of the VRP is commonly known as the general vehicle routing problem (GVRP). If the route length or range constraints are removed, then we refer to this problem as the standard vehicle routing problem (SVRP) (Bodin et al. 1983). We also define the time required for vehicle \( v \) to deliver or service at node \( i \) as \( s_i^v \), travel time for vehicle \( v \) from node \( i \) to node \( j \) as \( t_{ij}^v \), \( x_{ij}^v = 1 \) if arc \( i-j \) is used by vehicle \( v \) (\( x_{ij}^v = 0 \), otherwise), and \( c_{ij} \) as the cost of travelling from node \( i \) to node \( j \).

\[
\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{v=1}^{w} c_{ij} x_{ij}^v \quad (1)
\]

\[
\text{Subject to} \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^v = 1 \quad (j = 2,\ldots,n) \quad (2)
\]

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^v = 1 \quad (i = 2,\ldots,n) \quad (3)
\]

\[
\sum_{i=1}^{n} x_{ip}^v - \sum_{j=1}^{n} x_{pj}^v = 0 \quad (v = 1,\ldots,w; \ p = 1,\ldots,n) \quad (4)
\]

\[
\sum_{i=1}^{n} d_i (\sum_{j=1}^{n} x_{ij}^v) \leq K_v \quad (v = 1,\ldots,w) \quad (5)
\]

\[
\sum_{i=1}^{n} s_i^v \sum_{j=1}^{n} x_{ij}^v + \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij}^v x_{ij}^v \leq T_v \quad (v = 1,\ldots,w) \quad (6)
\]
\[
\sum_{j=2}^{n} x_{ij}^v \leq 1 \quad (v = 1,..,w)
\]

\[
\sum_{i=2}^{n} x_{ii}^v \leq 1 \quad (v = 1,..,w)
\]

\[X \in S \quad x_{ij}^v = 0 \text{ or } 1 \quad \text{for all } i, j, v\]

The objective function (1) minimizes the cost (travel distance) for all vehicles. Equations (2) and (3) ensure every customer is visited by one and only one vehicle. We assume that a customer’s demand does not exceed vehicle capacity and each customer is fully serviced by its one visiting vehicle. Equation (4) checks the continuity of our routes while (5) maintains the capacity constraint on all of the vehicles. Since we represent route length restrictions by time, (6) ensures maximum route times are not exceeded. Equations (7) and (8) insure we do not exceed vehicle fleet size. Next, let \(N^v \subseteq N\) represent the nodes from \(N\) assigned to vehicle \(v\) such that for any vehicle \(v\) that is not used, \(N^v = \emptyset\); \(N^1 \cup \ldots \cup N^{nv} = N\); and, \(N^1 \cap \ldots \cap N^{nv} = \emptyset\). The subtour breaking constraints are then defined and included in the model as

\[
x_{ij}^v : \sum_{i \in Q} \sum_{j \in Q} x_{ij} \geq 1 \text{ for every nonempty subset } Q \text{ of } N^v \quad \forall \ v = 1..nv.
\]

This states that for every proper subset \(Q\) of nodes must be connected to the other nodes in the network of the solution.

We eliminate some redundant constraints by recognizing that (2) and (4) enforces (3), while (4) and (7) imply (8) (Bodin et al. 1983).
Finally, we add time window considerations to the VRP. Let \( a_j \) represent the arrival time to node \( j \), \( e_j \) the earliest delivery time allowable, and \( l_j \) the no-later-than-time for delivery such that

\[
    a_j = \sum_i \sum_\nu (a_i + s_i^\nu + t_{ij}^\nu) x_{ij}^\nu \quad (j = 1, \ldots, n) \\
    a_I = 0 \\
    e_j \leq a_j \leq l_j \quad (j = 2, \ldots, n).
\]

For each \( j \), one of the \( x_{ij}^\nu \) variables equals 1, so \( a_j \) sums the previous arrival time \( (a_i) \), the service time at node \( i \) \( (s_i) \), and the travel time from \( i \) to \( j \) \( (t_{ij}) \). Alternatively, from Bodin et al. (1983), we can use the linear representation of time windows constraint in the formulation

\[
    \begin{align*}
    a_j &\geq (a_i + s_i^\nu + t_{ij}^\nu) - (1 - x_{ij}^\nu) T_{\text{max}}^\nu \quad \forall i, j, \nu \\
    a_j &\leq (a_i + s_i^\nu + t_{ij}^\nu) + (1 - x_{ij}^\nu) T_{\text{max}}^\nu
    \end{align*}
\]

When \( x_{ij}^\nu = 1 \), \( a_j \) is equal to the summation of the previous arrival time, previous service time and the travel time between the nodes. Conversely, when \( x_{ij}^\nu = 0 \) the constraints are redundant.

There are many alterations that could be added to this formulation to represent common real world problems. One such consideration takes into account the duty limitations of the crew that flies the vehicles. This can be done through inserting rest nodes that must be visited during the route that incur no travel cost, but impose service time equal to the mandatory rest break. Hard time windows for these rest nodes insure that the maximum duty hours will not be exceeded.
While the time windows defined in this formulation are hard, modeling the early time window as "soft" allows vehicles to arrive early, thus introducing a waiting time. Therefore, we use arrival times to calculate a waiting time that must be included in the precedence constraints along with service time and travel time.

Several changes are made to finalize the formulation of the strategic airlift system as modeled by MASS. First, we eliminate or soften time window constraints for the depots unless they are fixed and implement a version of the route length constraint (6) to insure route length limitations for a particular aircraft are not exceeded.

2.3 Methodology

The intent of this project is to explore the application of the Reactive Tabu Search (RTS) metaheuristic to routing problems, specifically the vehicle routing problem with time windows (VRPTW). This project has been coded in the object-oriented (OO) Java programming language for several reasons. First, the OO design of software allows us to reuse and modify existing code and libraries to reduce development time of new software. Second, Java programs are portable (Flanagan 1997). Finally, as an added benefit, the documentation tool, javadoc, links program documentation directly to the code for a hassle free method of updating and maintaining documentation. Javadoc extracts embedded comments in the code and creates an html file that is viewable with a web browser. This tool allows you to automatically create and maintain a single source file for accurate and useful documentation in the form of a web page (Eckel 1998).

The Java program represents a continuation of RTS code improvements starting with Carlton’s (1995) C code through Ryan et al. (1999) MODSIM implementation.
RTS follows the basic TS scheme but adjusts the tabu length based on the quality of the search, as determined by the number of iterations before a solution is revisited. (For example, a “high quality” search typically does not tend to revisit past solutions.) When the search moves to a neighbor solution that has been visited within the designated number of iterations or cycle length, the tabu length is increased by a multiplicative factor.

Conversely, if the solution has not been visited previously, tabu length is decreased by the multiplicative factor. When a solution is revisited within the maximum cycle length, a moving average of cycle lengths is calculated. If this average is less than the number of iterations without a change in tabu length, the current tabu length is decreased by the multiplicative factor. This concept from Battiti and Tecchiolli (1994) enforces the ultimate objective a broad exploration of the search space.

Finally, if all candidate solutions are tabu and aspiration criteria is not met, the search escapes to a solution with the smallest move value regardless of tabu status and then decreases the tabu length. This entire search routine is then continued for a designated number of iterations.

2.3.1 Tour Structure

The objective of the VRPTW is to find a tour in which each customer is visited within its stated time window by one vehicle, with a finite capacity, while minimizing the total cost. A tour is defined by the order in which the $n$ customers are served by the $m$ vehicles and is represented as an ordered list of the sequence of customers and vehicles, or “disjunctive graph” (Figure 3).
Figure 3. Disjunctive Graph representation of Tour

Positions 0 and $n + 1$ in this sequence represent depots, but are internally modeled as vehicles. Initially, the customers occupy positions between $I$ and $n_j$. Excess vehicles occupy positions after the last depot.

2.3.2 Starting Solution

Several methods are used to generate starting, but not necessarily feasible, solutions for the RTS algorithm. The time window “midpoint” is defined as halfway between the end service time (a no later than time) and the earlier begin service time (a no earlier than time) for a particular customer. In order starting tour (OST), we generate a “starting solution” by sorting the customers into an increasing time window midpoint value while enforcing the time window feasibility condition. Since the RTS is not limited to feasible starting solutions, the initial solution can sequentially read the initial list of customers (OST OFF), or this list can be randomly reordered and read to create a random-starting tour (RST) with a different staring point (and possibly an improved solution).
2.3.3 Solution Neighborhood

This search routine uses a disjunctive graph formulation internally to represent solution tours. From this representation the solution neighborhood is defined by the use of swap and insertion moves. A swap move is performed by exchanging the position of two adjacent nodes, while an insertion uses a series of successive swap moves to relocates a specific customer forwards or backwards in the tour by a number of steps called the insertion depth \((d)\).

Through the systematic use of these moves the RTS explores the vast solution space of the VRPTW. Starting with the initial solution, the algorithm searches insertion depths \(d \geq 1\) later in the tour (for customers 1 to \(n-1\)) and explores earlier insertions for depths \(d \leq 2\) (for customers 3 to \(n\)), comparing the candidate’s change in objective function from that of the incumbent tour. To reduce the vast set of candidate moves in a neighborhood, redundant tours are eliminated and the restriction of strong time window infeasibility is applied.

Redundant tours are tracked through the use of a two-attribute hashing scheme. The first attribute, hashing function \(f(T)\), is the objective function value \(Z(T)\). The second attribute, the tour hashing value \(\text{thv}\), takes the tour vector and calculates an integer value based on random integer values, \(\Psi(i)\), and the index of the customer assigned to tour position \(i\), \(\tau_i\) (Woodruff and Zemel 1993), such that

\[
\text{thv}(T) = \sum_{i=0}^{n} \Psi(\tau_i) \Psi(\tau_{i+1}).
\]

The tour hashing value attempts to minimize the possibility of a collision, or the incorrect identification of two tours as being identical or redundant when in fact they are distinct.
Additional attributes used to identify a solution are the tour cost, travel time, time window penalty, and total penalties. These integer values are concatenated in a string object that is uniquely identified in the Java programming language (java.util package) using the Hashtable class (Grand and Knudsen 1997). This unique numerical value is the “key” to identifying past solutions efficiently as well as accessing the “hash record”, where solution attributes are stored in their original form.

Strong window infeasibility states that whenever a vehicle leaves one node it can never arrive at the next node within its desired time window. Conversely, weak time window infeasible tours occur when only some departure times preclude a timely arrival at the next node. Unlike strong time window infeasibility, weak time window infeasible tours are evaluated in the search since insertion moves can ultimately reduce the amount of infeasibility in the overall tour. This is critical since past research has shown that feasible solution regions are isolated or disjoint in the solution space of these problems. In order to obtain an effective search, the method must investigate or accept infeasible solutions. This search of the infeasible region is facilitated through the use of penalty factors.

The ability to explore infeasible solutions represents a major advantage of this method for effectively exploring the solution space. First, instead of being restricted to regions of feasibility, RTS can traverse the regions of infeasibility to include using an infeasible initial solution. Second, the infeasible solutions recorded can be used in real world applications. For instance, an infeasible solution that produces very good results overall may become feasible with the relaxation of a constraint controlled by the decision-maker. These infeasible solutions represent the difficult choices faced by
managers trying to balance competing constraints when developing routes (this occurred in a delivery problem solved by Rochat and Semet in 1994).

From a solution neighborhood, the algorithm chooses the solution that results in the smallest move value. The move value is the difference between the incumbent tour’s objective function value and the candidate’s objective function value. The objective function value used in these initial tests includes change in travel time, change in waiting time, change in the time window penalty (lateness) and load penalty. With a relatively small amount of coding, the objective function can be expanded to include additional penalties, changed to represent several different weighted objective functions, or combined in a hierarchical objective function.

2.3.4 Tabu Criteria

Tabu search uses short-term memory to determine if a particular tour or attribute has already been visited by examining the attributes that comprise the tour. The examination must efficiently and reliably store and identify solution attributes previously visited during the search. We employ a “Tabulist” matrix of (n+1)*(n+1) dimensions with row numbers corresponding to customer identification number and columns corresponding to the index or position of the customer in the solution tour. The data elements in this array store a value equal to the iteration number that existed when the customer moved into this position, plus the tabu length. Later in the search, this value will be compared to the current iteration to determine if this attribute is tabu.
2.3.5 Algorithm Complexity

The size of the neighborhood considered at each step is $O(nd)$ and the computation of the move value for each neighbor is $O(n)$. If the depth of the insertion moves is restricted to 1, the algorithm achieves a computational complexity of $O(n^2)$. Thus, the worst case complexity is $O(n^2d)$, where $d$ is the depth of the allowable insertion moves. When the insertion depth is expanded, the computational complexity expands with it to $O(n^3)$. However, testing has shown empirically that considerably better times than $O(n^3)$ can be achieved, due to the strong time window infeasibility restriction (Carlton 1995).

1. Initialize starting variables ($k$ max iterations) and structures
2. Compute time matrix
3. Select starting tour
   a. Compute initial tour cost (Tour cost = Travel time + Penalty term)
   b. Compute initial hashing values
4. While ($k \leq$ niter)
   a. Look for incumbent tour in the hashing structure
      1) If found, update the iteration when found, increase the tabu length if applicable
      2) If not found, add to the hashing structure, decrease the tabu length, if applicable
   b. Evaluate all later insertions ($d \geq 1$, for customers 1 to n-1)
   c. Evaluate all earlier insertions ($d \leq -2$, for customers 3 to n)
   d. Move to the non-tabu neighbor. If all tours are tabu, move to the neighbor with the smallest move value, and reduce the tabu length.
   e. Update the search
      1) Incumbent tour schedule
      2) Incumbent tour hashing value
      3) Retain the best feasible solution found and the tour with the smallest tour cost regardless of feasibility
   f. $k = k + 1$
5. Output results

Figure 4. RTS Pseudocode (Carlton 1995)
2.4 Testing and Validation

Initial testing and validation uses the Solomon VRPTW/mTSPTW problem test set; specifically the 25, 50 and 100 customer problems with random, clustered, and random clustered distribution patterns. Computational results are compared to optimal answers obtained by Desrochers et al. (1992) (Tables 1-6). The first column identifies the problem instance. The second through fifth columns present the results obtained with the Java implemented RTS algorithm, i.e., the objective function value of minimum travel time, number of vehicles required, iteration of best feasible solution and the time (seconds) at which the solution was found, respectively. Similar information is presented in columns six through eight for the optimal solutions obtained by Desrochers et al. (1992). Columns 9 and 10 display the difference in travel time and the percentage difference between the optimal answer (when known) and the result obtained from the RTS algorithm. The last column shows the RTS starting method used to achieve the solution. OST is the ordered starting tour (arranged by time window midpoints). RST is the random arrangement of customers followed by the integer seed used. Listed order (LO) indicates that the initial solution is taken in exact order presented in the problem.

All problems were solved by the RTS algorithm using 2500 iterations, with an overall solution quality better than 99% of optimal in a fraction of the computational time required for the optimal solution. The increase in computational time from the mTSPTW algorithm to the VRPTW algorithm was negligible because most of the structure for the VRPTW was already included in the Java code for the mTSPTW algorithm.
<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(T)</th>
<th>Iter</th>
<th>Time</th>
<th>Z(T)</th>
<th>Iter</th>
<th>Time</th>
<th>Difference Δ</th>
<th>Δ%</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>867.1</td>
<td>8</td>
<td>317</td>
<td>3</td>
<td>867.1</td>
<td>8</td>
<td>5.8</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R102</td>
<td>797.1</td>
<td>7</td>
<td>35</td>
<td>1</td>
<td>797.1</td>
<td>7</td>
<td>20.3</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R103</td>
<td>704.6</td>
<td>5</td>
<td>132</td>
<td>1</td>
<td>704.6</td>
<td>5</td>
<td>22.2</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R104</td>
<td>666.9</td>
<td>4</td>
<td>86</td>
<td>1</td>
<td>666.9</td>
<td>4</td>
<td>46.0</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R105</td>
<td>780.5</td>
<td>6</td>
<td>95</td>
<td>1</td>
<td>780.5</td>
<td>6</td>
<td>22.6</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R106</td>
<td>715.4</td>
<td>5</td>
<td>28</td>
<td>0</td>
<td>715.4</td>
<td>5</td>
<td>205.2</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R107</td>
<td>674.3</td>
<td>4</td>
<td>2080</td>
<td>23</td>
<td>674.3</td>
<td>4</td>
<td>304.1</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R108</td>
<td>647.3</td>
<td>4</td>
<td>45</td>
<td>0</td>
<td>647.3</td>
<td>4</td>
<td>307.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R109</td>
<td>691.3</td>
<td>5</td>
<td>21</td>
<td>0</td>
<td>691.3</td>
<td>5</td>
<td>14.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R110</td>
<td>694.1</td>
<td>5</td>
<td>91</td>
<td>2</td>
<td>679.8</td>
<td>4</td>
<td>64.3</td>
<td>14.3</td>
<td>2.1%</td>
</tr>
<tr>
<td>R111</td>
<td>678.8</td>
<td>4</td>
<td>178</td>
<td>2</td>
<td>678.8</td>
<td>4</td>
<td>330.3</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>R112</td>
<td>643.0</td>
<td>4</td>
<td>25</td>
<td>0</td>
<td>643.0</td>
<td>4</td>
<td>623.3</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C101</td>
<td>2441.3</td>
<td>3</td>
<td>23</td>
<td>0</td>
<td>2441.3</td>
<td>3</td>
<td>18.6</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C102</td>
<td>2440.3</td>
<td>3</td>
<td>379</td>
<td>4</td>
<td>2440.3</td>
<td>3</td>
<td>79.9</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C103</td>
<td>2440.3</td>
<td>3</td>
<td>72</td>
<td>1</td>
<td>2440.3</td>
<td>3</td>
<td>134.7</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C104</td>
<td>2436.9</td>
<td>3</td>
<td>797</td>
<td>8</td>
<td>2436.9</td>
<td>3</td>
<td>223.9</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C105</td>
<td>2441.3</td>
<td>3</td>
<td>209</td>
<td>2</td>
<td>2441.3</td>
<td>3</td>
<td>25.6</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C106</td>
<td>2441.3</td>
<td>3</td>
<td>26</td>
<td>1</td>
<td>2441.3</td>
<td>3</td>
<td>20.7</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C107</td>
<td>2441.3</td>
<td>3</td>
<td>28</td>
<td>1</td>
<td>2441.3</td>
<td>3</td>
<td>31.7</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C108</td>
<td>2441.3</td>
<td>3</td>
<td>1421</td>
<td>15</td>
<td>2441.3</td>
<td>3</td>
<td>43.1</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>C109</td>
<td>2441.3</td>
<td>3</td>
<td>148</td>
<td>1</td>
<td>2441.3</td>
<td>3</td>
<td>585.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC101</td>
<td>711.1</td>
<td>4</td>
<td>214</td>
<td>3</td>
<td>711.1</td>
<td>4</td>
<td>225.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC102</td>
<td>601.7</td>
<td>3</td>
<td>20</td>
<td>1</td>
<td>596.0</td>
<td>3</td>
<td>18.1</td>
<td>5.7</td>
<td>0.96%</td>
</tr>
<tr>
<td>RC103</td>
<td>582.8</td>
<td>3</td>
<td>2193</td>
<td>24</td>
<td>582.8</td>
<td>3</td>
<td>103.0</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC104</td>
<td>556.6</td>
<td>3</td>
<td>604</td>
<td>6</td>
<td>556.6</td>
<td>3</td>
<td>177.9</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC105</td>
<td>661.2</td>
<td>4</td>
<td>79</td>
<td>1</td>
<td>661.2</td>
<td>4</td>
<td>37.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC106</td>
<td>595.5</td>
<td>3</td>
<td>60</td>
<td>1</td>
<td>595.5</td>
<td>3</td>
<td>248.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC107</td>
<td>548.3</td>
<td>3</td>
<td>69</td>
<td>1</td>
<td>548.3</td>
<td>3</td>
<td>113.9</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>RC108</td>
<td>544.5</td>
<td>3</td>
<td>2203</td>
<td>23</td>
<td>544.5</td>
<td>3</td>
<td>256.0</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>Average</td>
<td>1218.19</td>
<td>3.93</td>
<td>402.7</td>
<td>4.38</td>
<td>1184.8</td>
<td>3.90</td>
<td>148.6</td>
<td>0.69</td>
<td>0.11%</td>
</tr>
</tbody>
</table>

1. Maximum number of vehicles: m = 10. Time window penalty: \( p_{TW} = 1.0 \).
2. Maximum iterations: \( k = 2500 \).
3. Seconds on a Pentium II 400 MHz system. Total runtime \approx 28 seconds each.
4. Seconds on a SUN SPARK 1.
5. OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O’Rourke 1999)
Table 2. Solomon mTSPTW (50 Customers)

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Ryer &amp; O’Rourke</th>
<th></th>
<th></th>
<th>Optimal</th>
<th></th>
<th></th>
<th>Difference</th>
<th>Start Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z(T)</td>
<td>Used</td>
<td>Iter</td>
<td>Time</td>
<td>Z(T)</td>
<td>Used</td>
<td>Time</td>
<td>Δ</td>
</tr>
<tr>
<td>R101</td>
<td>1543.8</td>
<td>12</td>
<td>239</td>
<td>9</td>
<td>1535.2</td>
<td>12</td>
<td>66.7</td>
<td>8.6</td>
</tr>
<tr>
<td>R102</td>
<td>1409.0</td>
<td>11</td>
<td>1939</td>
<td>78</td>
<td>1404.6</td>
<td>11</td>
<td>67.8</td>
<td>4.4</td>
</tr>
<tr>
<td>R103</td>
<td>1282.7</td>
<td>9</td>
<td>871</td>
<td>36</td>
<td>1272.3</td>
<td>9</td>
<td>8939.1</td>
<td>10.2</td>
</tr>
<tr>
<td>R104</td>
<td>1131.9</td>
<td>6</td>
<td>734</td>
<td>31</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>R105</td>
<td>1401.6</td>
<td>9</td>
<td>402</td>
<td>15</td>
<td>1399.2</td>
<td>9</td>
<td>362.6</td>
<td>2.4</td>
</tr>
<tr>
<td>R106</td>
<td>1293.0</td>
<td>8</td>
<td>2294</td>
<td>94</td>
<td>1285.2</td>
<td>8</td>
<td>386.4</td>
<td>7.8</td>
</tr>
<tr>
<td>R107</td>
<td>1211.1</td>
<td>7</td>
<td>1786</td>
<td>75</td>
<td>1211.1</td>
<td>7</td>
<td>7362.1</td>
<td>0.0</td>
</tr>
<tr>
<td>R108</td>
<td>1117.7</td>
<td>6</td>
<td>1698</td>
<td>75</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>R109</td>
<td>1286.7</td>
<td>8</td>
<td>1452</td>
<td>58</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>R110</td>
<td>1207.8</td>
<td>7</td>
<td>1853</td>
<td>78</td>
<td>1197.0</td>
<td>7</td>
<td>4906.1</td>
<td>10.8</td>
</tr>
<tr>
<td>R111</td>
<td>1216.6</td>
<td>7</td>
<td>1775</td>
<td>72</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>R112</td>
<td>1140.5</td>
<td>6</td>
<td>1784</td>
<td>78</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C101</td>
<td>4862.4</td>
<td>5</td>
<td>119</td>
<td>4</td>
<td>4862.4</td>
<td>5</td>
<td>67.1</td>
<td>0.0</td>
</tr>
<tr>
<td>C102</td>
<td>4861.4</td>
<td>5</td>
<td>607</td>
<td>19</td>
<td>4861.4</td>
<td>5</td>
<td>330.2</td>
<td>0.0</td>
</tr>
<tr>
<td>C103</td>
<td>4855.8</td>
<td>5</td>
<td>1699</td>
<td>57</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C104</td>
<td>4884.1</td>
<td>5</td>
<td>1253</td>
<td>43</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C105</td>
<td>4861.2</td>
<td>5</td>
<td>232</td>
<td>7</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C106</td>
<td>4862.4</td>
<td>5</td>
<td>308</td>
<td>9</td>
<td>4862.4</td>
<td>5</td>
<td>91.3</td>
<td>0.0</td>
</tr>
<tr>
<td>C107</td>
<td>4861.2</td>
<td>5</td>
<td>382</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C108</td>
<td>4861.2</td>
<td>5</td>
<td>92</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C109</td>
<td>4860.9</td>
<td>5</td>
<td>301</td>
<td>9</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC101</td>
<td>1444.0</td>
<td>8</td>
<td>1252</td>
<td>38</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC102</td>
<td>1325.1</td>
<td>7</td>
<td>754</td>
<td>23</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC103</td>
<td>1216.2</td>
<td>6</td>
<td>1589</td>
<td>54</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC104</td>
<td>1046.5</td>
<td>5</td>
<td>860</td>
<td>31</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC105</td>
<td>1355.3</td>
<td>8</td>
<td>248</td>
<td>8</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC106</td>
<td>1223.2</td>
<td>6</td>
<td>1921</td>
<td>61</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC107</td>
<td>1146.0</td>
<td>6</td>
<td>189</td>
<td>7</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RC108</td>
<td>1098.1</td>
<td>6</td>
<td>1821</td>
<td>65</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Average</td>
<td>2374.7</td>
<td>6.66</td>
<td>1050</td>
<td>39.6</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

1 Maximum number of vehicles: R sets m = 15; C sets m = 6; RC sets m = 8. Time window penalty: $\rho_{tw} = 3.0$.
2 Maximum iterations: $k = 2500$.
3 Seconds on a Pentium II 400 MHz system. Total runtime ~ 100 seconds each.
4 Seconds on a SUN SPARK 1.
5 OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O’Rourke 1999)
**Table 3. Solomon mTSPTW (100 Customers)**

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(7)</th>
<th>Used</th>
<th>Iter</th>
<th>Time</th>
<th>Optimal</th>
<th>Z(7)</th>
<th>Used</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>2689.6</td>
<td>20</td>
<td>2167</td>
<td>371</td>
<td>2607.7</td>
<td>18</td>
<td>1064.2</td>
<td></td>
</tr>
<tr>
<td>R102</td>
<td>2522.9</td>
<td>18</td>
<td>1783</td>
<td>322</td>
<td>2434.0</td>
<td>17</td>
<td>756.9</td>
<td></td>
</tr>
<tr>
<td>R103</td>
<td>2266.8</td>
<td>15</td>
<td>1797</td>
<td>351</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R104</td>
<td>2010.6</td>
<td>11</td>
<td>1401</td>
<td>311</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R105</td>
<td>2418.0</td>
<td>16</td>
<td>560</td>
<td>93</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R106</td>
<td>2256.9</td>
<td>14</td>
<td>1403</td>
<td>252</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R107</td>
<td>2091.6</td>
<td>12</td>
<td>1462</td>
<td>278</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R108</td>
<td>1980.3</td>
<td>10</td>
<td>2325</td>
<td>491</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R109</td>
<td>2191.4</td>
<td>13</td>
<td>2149</td>
<td>398</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R110</td>
<td>2121.1</td>
<td>12</td>
<td>1479</td>
<td>291</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R111</td>
<td>2082.1</td>
<td>12</td>
<td>1882</td>
<td>370</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>R112</td>
<td>1986.1</td>
<td>11</td>
<td>2325</td>
<td>507</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

| C101     | 9827.3 | 10   | 285  | 45   | 9827.3  | 10   | 434.5 |
| C102     | 9820.3 | 10   | 237  | 42   | —       | —    | —     |
| C103     | 9813.7 | 10   | 256  | 49   | —       | —    | —     |
| C104     | 9809.0 | 10   | 2495 | 536  | —       | —    | —     |
| C105     | 9821.2 | 10   | 313  | 50   | —       | —    | —     |
| C106     | 9827.3 | 10   | 455  | 75   | 9827.3  | 10   | 724.8 |
| C107     | 9818.9 | 10   | 292  | 48   | —       | —    | —     |
| C108     | 9818.9 | 10   | 662  | 115  | —       | —    | —     |
| C109     | 9818.6 | 10   | 1381 | 262  | —       | —    | —     |

| RC101    | 2685.7 | 16   | 897  | 144  | —       | —    | —     |
| RC102    | 2534.0 | 15   | 2410 | 434  | —       | —    | —     |
| RC103    | 2352.3 | 13   | 1047 | 195  | —       | —    | —     |
| RC104    | 2209.1 | 11   | 1311 | 272  | —       | —    | —     |
| RC105    | 2538.0 | 15   | 2327 | 412  | —       | —    | —     |
| RC106    | 2457.8 | 14   | 443  | 74   | —       | —    | —     |
| RC107    | 2236.9 | 12   | 1822 | 344  | —       | —    | —     |
| RC108    | 2115.9 | 11   | 2206 | 451  | —       | —    | —     |

| Average  | 4624.9 | 12.45| 1365 | 261.48| —       | —    | —     |

1 Maximum number of vehicles: $m = 25$. Time window penalty: $\rho_w = 8.0$.
2 Maximum iterations: $k = 2500$.
3 Seconds on a Pentium II 400 MHz system. Total runtime ~ 550 seconds each.
4 Seconds on a SUN SPARK 1.
5 OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O'Rourke 1999)
## Table 4. Solomon VRPTW (25 Customers)

<table>
<thead>
<tr>
<th>Prob Set$^1$</th>
<th>Lower &amp; O’Rourke</th>
<th>Optimal</th>
<th>Difference</th>
<th>Start Method$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z(7) Used</td>
<td>Iter$^3$</td>
<td>Time$^4$</td>
<td>Z(7) Used</td>
</tr>
<tr>
<td>R101</td>
<td>867.1</td>
<td>8</td>
<td>317</td>
<td>4</td>
</tr>
<tr>
<td>R102</td>
<td>797.1</td>
<td>7</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>R103</td>
<td>704.6</td>
<td>5</td>
<td>132</td>
<td>1</td>
</tr>
<tr>
<td>R104</td>
<td>666.9</td>
<td>4</td>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td>R105</td>
<td>780.5</td>
<td>6</td>
<td>95</td>
<td>1</td>
</tr>
<tr>
<td>R106</td>
<td>715.4</td>
<td>5</td>
<td>1149</td>
<td>12</td>
</tr>
<tr>
<td>R107</td>
<td>674.3</td>
<td>4</td>
<td>2080</td>
<td>24</td>
</tr>
<tr>
<td>R108</td>
<td>647.3</td>
<td>4</td>
<td>58</td>
<td>1</td>
</tr>
<tr>
<td>R109</td>
<td>691.3</td>
<td>5</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>R110</td>
<td>694.1</td>
<td>5</td>
<td>91</td>
<td>1</td>
</tr>
<tr>
<td>R111</td>
<td>678.8</td>
<td>4</td>
<td>178</td>
<td>3</td>
</tr>
<tr>
<td>R112</td>
<td>643.0</td>
<td>4</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>C101</td>
<td>2441.3</td>
<td>3</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>C102</td>
<td>2440.3</td>
<td>3</td>
<td>106</td>
<td>1</td>
</tr>
<tr>
<td>C103</td>
<td>2440.3</td>
<td>3</td>
<td>72</td>
<td>1</td>
</tr>
<tr>
<td>C104</td>
<td>2436.9</td>
<td>3</td>
<td>741</td>
<td>8</td>
</tr>
<tr>
<td>C105</td>
<td>2441.3</td>
<td>3</td>
<td>170</td>
<td>1</td>
</tr>
<tr>
<td>C106</td>
<td>2441.3</td>
<td>3</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>C107</td>
<td>2441.3</td>
<td>3</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>C108</td>
<td>2441.3</td>
<td>3</td>
<td>455</td>
<td>4</td>
</tr>
<tr>
<td>C109</td>
<td>2441.3</td>
<td>3</td>
<td>197</td>
<td>2</td>
</tr>
<tr>
<td>RC101</td>
<td>711.1</td>
<td>4</td>
<td>214</td>
<td>2</td>
</tr>
<tr>
<td>RC102</td>
<td>601.7</td>
<td>3</td>
<td>149</td>
<td>1</td>
</tr>
<tr>
<td>RC103</td>
<td>582.8</td>
<td>3</td>
<td>134</td>
<td>2</td>
</tr>
<tr>
<td>RC104</td>
<td>556.6</td>
<td>3</td>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td>RC105</td>
<td>661.2</td>
<td>4</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>RC106</td>
<td>595.5</td>
<td>3</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>RC107</td>
<td>548.3</td>
<td>3</td>
<td>179</td>
<td>2</td>
</tr>
<tr>
<td>RC108</td>
<td>544.5</td>
<td>3</td>
<td>353</td>
<td>3</td>
</tr>
</tbody>
</table>

Average 1218.2 3.93 250.7 2.86 1184.8 3.90 148.6 0.69 0.11% LO

---

1 Maximum number of vehicles: $m = 10$. Time window penalty: $p_{TW} = 8.0$; load penalty $p_{LO} = 10.0$.
2 Maximum iterations: $k = 2500$.
3 Seconds on a Pentium II 400 MHz system. Total runtime ~ 28 seconds each.
4 Seconds on a SUN SPARK 1.
5 OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O’Rourke 1999)
<table>
<thead>
<tr>
<th>Prob Set¹</th>
<th>Ruyer &amp; O'Rourke Z((T)) Used</th>
<th>Iter²</th>
<th>Time³</th>
<th>Optimal Z((T)) Used</th>
<th>Time³</th>
<th>Difference Δ</th>
<th>Δ%</th>
<th>Method⁵</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>1543.8</td>
<td>12</td>
<td>239</td>
<td>9</td>
<td>1535.2</td>
<td>12</td>
<td>66.7</td>
<td>8.6</td>
</tr>
<tr>
<td>R102</td>
<td>1409.0</td>
<td>11</td>
<td>1939</td>
<td>82</td>
<td>1404.6</td>
<td>11</td>
<td>67.8</td>
<td>4.4</td>
</tr>
<tr>
<td>R103</td>
<td>1278.7</td>
<td>9</td>
<td>1935</td>
<td>87</td>
<td>1272.5</td>
<td>9</td>
<td>8939.1</td>
<td>6.2</td>
</tr>
<tr>
<td>R104</td>
<td>1137.4</td>
<td>6</td>
<td>1533</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R105</td>
<td>1401.6</td>
<td>9</td>
<td>402</td>
<td>16</td>
<td>1399.2</td>
<td>9</td>
<td>362.6</td>
<td>2.4</td>
</tr>
<tr>
<td>R106</td>
<td>1293.0</td>
<td>8</td>
<td>2294</td>
<td>99</td>
<td>1285.2</td>
<td>8</td>
<td>386.4</td>
<td>7.8</td>
</tr>
<tr>
<td>R107</td>
<td>1211.1</td>
<td>7</td>
<td>1786</td>
<td>79</td>
<td>1211.1</td>
<td>7</td>
<td>7362.1</td>
<td>0.0</td>
</tr>
<tr>
<td>R108</td>
<td>1117.7</td>
<td>6</td>
<td>1698</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R109</td>
<td>1286.7</td>
<td>8</td>
<td>1451</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R110</td>
<td>1207.8</td>
<td>7</td>
<td>1853</td>
<td>84</td>
<td>1197.0</td>
<td>7</td>
<td>4906.1</td>
<td>10.8</td>
</tr>
<tr>
<td>R111</td>
<td>1216.6</td>
<td>7</td>
<td>1775</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R112</td>
<td>1135.0</td>
<td>6</td>
<td>1456</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C101</td>
<td>4862.4</td>
<td>5</td>
<td>74</td>
<td>3</td>
<td>4862.4</td>
<td>5</td>
<td>67.1</td>
<td>0.0</td>
</tr>
<tr>
<td>C102</td>
<td>4861.4</td>
<td>5</td>
<td>232</td>
<td>9</td>
<td>4861.4</td>
<td>5</td>
<td>330.2</td>
<td>0.0</td>
</tr>
<tr>
<td>C103</td>
<td>4861.4</td>
<td>5</td>
<td>2035</td>
<td>87</td>
<td>4861.4</td>
<td>5</td>
<td>896.0</td>
<td>0.0</td>
</tr>
<tr>
<td>C104</td>
<td>4882.8</td>
<td>5</td>
<td>1727</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C105</td>
<td>4862.4</td>
<td>5</td>
<td>494</td>
<td>19</td>
<td>4862.4</td>
<td>5</td>
<td>99.1</td>
<td>0.0</td>
</tr>
<tr>
<td>C106</td>
<td>4862.4</td>
<td>5</td>
<td>91</td>
<td>4</td>
<td>4862.4</td>
<td>5</td>
<td>91.3</td>
<td>0.0</td>
</tr>
<tr>
<td>C107</td>
<td>4862.4</td>
<td>5</td>
<td>154</td>
<td>6</td>
<td>4862.4</td>
<td>5</td>
<td>170.6</td>
<td>0.0</td>
</tr>
<tr>
<td>C108</td>
<td>4862.4</td>
<td>5</td>
<td>95</td>
<td>4</td>
<td>4862.4</td>
<td>5</td>
<td>245.6</td>
<td>0.0</td>
</tr>
<tr>
<td>C109</td>
<td>4862.4</td>
<td>5</td>
<td>643</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC101</td>
<td>1446.8</td>
<td>8</td>
<td>1613</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC102</td>
<td>1331.8</td>
<td>7</td>
<td>1508</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC103</td>
<td>1210.9</td>
<td>6</td>
<td>2194</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC104</td>
<td>1046.5</td>
<td>5</td>
<td>412</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC105</td>
<td>1355.3</td>
<td>8</td>
<td>104</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC106</td>
<td>1223.2</td>
<td>6</td>
<td>1454</td>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC107</td>
<td>1144.4</td>
<td>6</td>
<td>898</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC108</td>
<td>1098.1</td>
<td>6</td>
<td>1361</td>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>2375.01</td>
<td>6.66</td>
<td>1153</td>
<td>49.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ Maximum number of vehicles: \(m = 15\). Time window penalty: \(p_{TW} = 1.0\); load penalty \(p_{LO} = 10.0\).
² Maximum iterations \(k = 2500\).
³ Seconds on a Pentium II 400 MHz system. Total runtime \(\sim 100\) seconds each.
⁴ Seconds on a SUN SPARK 1.
⁵ OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O’Rourke 1999)
## Table 6. Solomon VRPTW (100 Customers)

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(T)</th>
<th>Used</th>
<th>Iter</th>
<th>Time</th>
<th>Z(T)</th>
<th>Used</th>
<th>Time</th>
<th>Δ</th>
<th>Δ%</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>2676.2</td>
<td>20</td>
<td>2271</td>
<td>414</td>
<td>2607.7</td>
<td>18</td>
<td>1064.2</td>
<td>68.5</td>
<td>2.63%</td>
<td>RST 2</td>
</tr>
<tr>
<td>R102</td>
<td>2502.4</td>
<td>19</td>
<td>492</td>
<td>96</td>
<td>2434.0</td>
<td>17</td>
<td>756.9</td>
<td>68.4</td>
<td>2.81%</td>
<td>RST 0</td>
</tr>
<tr>
<td>R103</td>
<td>2265.0</td>
<td>15</td>
<td>1091</td>
<td>228</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 2</td>
</tr>
<tr>
<td>R104</td>
<td>2039.6</td>
<td>12</td>
<td>1488</td>
<td>338</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R105</td>
<td>2399.4</td>
<td>16</td>
<td>1974</td>
<td>378</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R106</td>
<td>2268.4</td>
<td>14</td>
<td>2431</td>
<td>491</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R107</td>
<td>2129.0</td>
<td>13</td>
<td>1905</td>
<td>406</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R108</td>
<td>1956.8</td>
<td>10</td>
<td>2415</td>
<td>565</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R109</td>
<td>2181.0</td>
<td>14</td>
<td>1587</td>
<td>311</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R110</td>
<td>2133.2</td>
<td>13</td>
<td>1548</td>
<td>328</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R111</td>
<td>2077.3</td>
<td>12</td>
<td>2248</td>
<td>491</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>R112</td>
<td>1971.6</td>
<td>11</td>
<td>1898</td>
<td>460</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(T)</th>
<th>Used</th>
<th>Iter</th>
<th>Time</th>
<th>Z(T)</th>
<th>Used</th>
<th>Time</th>
<th>Δ</th>
<th>Δ%</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>C101</td>
<td>9827.3</td>
<td>10</td>
<td>263</td>
<td>43</td>
<td>9827.3</td>
<td>10</td>
<td>434.5</td>
<td>0.0</td>
<td>0.00%</td>
<td>OST</td>
</tr>
<tr>
<td>C102</td>
<td>9827.3</td>
<td>10</td>
<td>1317</td>
<td>253</td>
<td>9827.3</td>
<td>10</td>
<td>1990.8</td>
<td>0.0</td>
<td>0.00%</td>
<td>OST</td>
</tr>
<tr>
<td>C103</td>
<td>9828.9</td>
<td>10</td>
<td>2500</td>
<td>535</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>C104</td>
<td>9949.6</td>
<td>10</td>
<td>2194</td>
<td>509</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>C105</td>
<td>9827.3</td>
<td>10</td>
<td>378</td>
<td>65</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>C106</td>
<td>9827.3</td>
<td>10</td>
<td>309</td>
<td>55</td>
<td>9827.3</td>
<td>10</td>
<td>724.8</td>
<td>0.0</td>
<td>0.00%</td>
<td>OST</td>
</tr>
<tr>
<td>C107</td>
<td>9827.3</td>
<td>10</td>
<td>1144</td>
<td>210</td>
<td>9827.3</td>
<td>10</td>
<td>1010.4</td>
<td>0.0</td>
<td>0.00%</td>
<td>OST</td>
</tr>
<tr>
<td>C108</td>
<td>9827.3</td>
<td>10</td>
<td>1638</td>
<td>321</td>
<td>9827.3</td>
<td>10</td>
<td>1613.6</td>
<td>0.0</td>
<td>0.00%</td>
<td>OST</td>
</tr>
<tr>
<td>C109</td>
<td>9853.3</td>
<td>10</td>
<td>2202</td>
<td>463</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(T)</th>
<th>Used</th>
<th>Iter</th>
<th>Time</th>
<th>Z(T)</th>
<th>Used</th>
<th>Time</th>
<th>Δ</th>
<th>Δ%</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC101</td>
<td>2669.9</td>
<td>16</td>
<td>2110</td>
<td>381</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>OST</td>
</tr>
<tr>
<td>RC102</td>
<td>2498.4</td>
<td>15</td>
<td>2136</td>
<td>419</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC103</td>
<td>2363.6</td>
<td>13</td>
<td>1333</td>
<td>270</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC104</td>
<td>2179.2</td>
<td>11</td>
<td>1365</td>
<td>308</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC105</td>
<td>2557.4</td>
<td>15</td>
<td>2482</td>
<td>473</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC106</td>
<td>2432.8</td>
<td>13</td>
<td>2222</td>
<td>434</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC107</td>
<td>2266.1</td>
<td>12</td>
<td>2024</td>
<td>417</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
<tr>
<td>RC108</td>
<td>2175.1</td>
<td>12</td>
<td>2122</td>
<td>475</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>RST 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prob Set</th>
<th>Z(T)</th>
<th>Used</th>
<th>Iter</th>
<th>Time</th>
<th>Z(T)</th>
<th>Used</th>
<th>Time</th>
<th>Δ</th>
<th>Δ%</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4632.3</td>
<td>12.62</td>
<td>1693</td>
<td>349.6</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

1 Maximum number of vehicles: m = 25. Time window penalty: ρ_{TW} = 8.0; load penalty ρ_{LO} = 10.0.
2 Maximum iterations k = 2500.
3 Seconds on a Pentium II 400 MHz system. Total runtime ~ 550 seconds each.
4 Seconds on a SUN SPARK 1.
5 OST is ordered starting tour. RST # is random starting tour where # is the seed value. LO is listed ordering.

(O'Rourke 1999)
2.5 Extensions for Solving Mobility Routing Problems

The first step in transforming this algorithm from solving academic test problems to tackling global routing problems is transitioning from the x-y plane to geographic coordinates. This is accomplished in conjunction with the ability to determine an aircraft’s groundspeed based on its associated true airspeed, the prevalent wind direction and speed.

To incorporate the effect of winds on the RTS algorithm, the distance and bearing is first calculated as shown in Departments of the Air Force and Navy’s AFR 51-40. Given the departure latitude \((L_1)\) and longitude \((\lambda_1)\) and destination latitude \((L_2)\) and longitude \((\lambda_2)\), the great circle distance in nautical miles \((D)\) can be found using the following formulation.

\[
D = 60 \times \cos^{-1} \left[ \sin L_1 \times \sin L_2 + \cos L_1 \times \cos L_2 \times \cos (\lambda_2 - \lambda_1) \right]
\]

Using this distance, the heading angle \((H)\) in degrees is

\[
H = \cos^{-1} \left[ \frac{\sin L_2 - \sin L_1 \times \cos(D/60)}{\sin(D/60) \times \cos \lambda_1} \right].
\]

Correcting this angle to the proper quadrant the initial true heading \((\Theta_{XY})\) is

\[
\Theta_{XY} = H \text{ if } \sin (\lambda_2 - \lambda_1) < 0
\]

or

\[
\Theta_{XY} = 360 - H \text{ if } \sin (\lambda_2 - \lambda_1) \geq 0.
\]

Finally, using the bearing from the departure point to the destination point, current airspeed, wind speed and direction, a ground speed can be calculated. The true heading of the wind is represented by \(\Theta_{WS}\) and the course offset from true heading from \(X\) to \(Y\) is
denoted by \( \gamma \), thus adjusting heading for the wind drift. When the wind direction results in a headwind component, the angle between \( \Theta_{XY} \) and \( \Theta_{WS} (\delta) \) is less than 90 degrees. The wind component of the groundspeed \( (A) \) becomes negative and thus reduces the overall groundspeed. Conversely, when winds result in a tailwind component, \( \delta \) is greater than 90 degrees (Figure 5), \( A \) becomes positive and increases the overall groundspeed.

\[
\cos(180 - \delta) = \frac{A}{WS} \\
A = WS \cdot \cos(180 - \delta)
\]

\[
\sin(180 - \delta) = \frac{C}{WS} \\
C = WS \cdot \sin(180 - \delta)
\]

\[
B^2 + C^2 = A S^2 \\
B = \sqrt{A S^2 - C^2}
\]

\[
GS = A + B \\
GS = WS \cdot \cos(180 - \delta) + \sqrt{A S^2 - WS^2 \cdot \sin^2(180 - \delta)}
\]

Figure 5. Calculation of Groundspeed to Account for Winds

With the translation to a real-world geographical coordinate representation complete, the time matrix used by the RTS can be updated to tackle realistic routing problems.
The second algorithm extension adds a restriction on possible routes based on the maximum leg distance for a vehicle. A simple search of those time matrix values that exceed the maximum leg distance, with the subsequent substitution of a very large value for such elements of the time matrix, precludes the RTS from selecting those routes in its final solution (unless no other feasible solution exists). In a similar manner we can restrict prohibited international flight routes by accessing the time matrix directly and assigning the same large value.

The next critical dimension of handling a global routing problem is an extension to handle multi-depot problems. This required additional logic to ensure vehicles assigned to different depots return to their respective starting depot while accounting for the change in travel time based on their respective depot location. Each vehicle node can be considered an aggregation of a return node for the previous vehicle and a start node for the next vehicle with zero cost between the two. Cost calculation “into” the node is assigned the distance from the customer back to its appropriate depot. This is allowed since multi-depot position integrity is maintained due the fact that vehicle ordering is strictly enforced by the algorithm. As implemented in this algorithm, multiple depots are modeled with the restriction that available vehicles be assigned to desired depots at the onset. Regardless of depot location, only those vehicles resulting in the best solution will be chosen.

2.6 Dimensions of the Mobility Routing Problem

When MASS flight plans a route, it evaluates the feasibility of fuel requirements, allowable cabin load (ACL), maximum on ground (MOG) feasibility, crew duty day
(CDD), and then updates the crew plan. This Aircraft Flight Plan Algorithm (AFPA) is capable of incorporating many of these considerations when selecting the routes during its search. Vehicle fuel requirement is implemented by the maximum leg restriction discussed earlier. Capacity constraints that represent the VRPTW can be extended in the future to include precedence for a pickup and delivery problem (PDP) version if required. Currently, MOG is captured by service and wait times. Crew duty day (not currently implemented) can be tracked by individual aircraft (since crews are modeled in MASS as a single entity), with a CDD clock refreshed either through mandatory waiting times or at bases that have rested crews available.

Time windows become necessary for two reasons. First, every AMC airlift scenario uses a document known as the Time-Phased Force Deployment Data (TPFDD) document, which specifies origins, destinations, cargo types, and their required delivery times (Cox 1998). Although these delivery times define the “not-earlier-than-time” (NET) and the “not-later-than-time” (NLT) for each individual piece of cargo, the NET and NLT can be used to derive the time window boundaries for origins and destinations. Time windows can also take into account normal operating hours of bases that are subject to the constraint of quiet hours.

Finally, the apparent problem of applying a VRP format (all destinations must be visited once – no more, no less) to an aircraft routing problem can be overcome with the inclusion of multiple customers at the same location. These customers share the total demand between them and may or may not have similar time windows. The algorithm determines the number of aircraft needed to service these destinations based on the capacity requirements and time window restrictions. Air refueling points are
incorporated as customers, with zero demand and service time, at strategic locations in the scenario, or at established air refueling tracks.

All these considerations are important and have a direct impact on route selection. Ignoring them and determining routes based on distance alone can not accurately represent the best routes needed for MASS. By starting with a simple scenario and adding these constraints, the effect on route selection becomes readily apparent. Deterministic approaches often reach their limit in ability to solve very basic VRPTW's when their size exceeds 50 customers.

The computational advantages of solving a real world problem, as well as the effect of additional constraints on route selection are shown using a notional problem involving the 50 United States capitals. A good feasible solution was obtained in less than 30 seconds on a Dell 266 Pentium II lap top computer (Figure 6). With the additional considerations of time windows, servicing, capacity and the possible use of multiple depots, the solution to the 50 U. S. capitals problem is altered dramatically (Figure 7).

The algorithm presented thus far uses a reactive tabu length to intensify and diversify the search. With the expansion of the algorithm to include additional constraints, the need for reactive penalty functions becomes essential. Reactive penalty functions presented by Gendreau and Laporte (1996) offer the benefit of incorporating reactive penalty parameters in their RTS algorithm. The penalty coefficients are set at an initial value $\rho$ and then multiplied every ten iterations by $2^{(t/5)-1}$, where $t$ is the number of feasible solutions among the last ten solutions. Based on the number of feasible solutions $\rho$ is either increased or decreased accordingly. The resulting mix of feasible and
Figure 6. Solution of Simple TSP comprised of the 50 U. S. Capitals
Relaxed Scenario
(1 vehicle required)

Time Window and Service Time Constraints
(4 vehicles required)

Capacity Constraints
(5 vehicles required)

Multiple Depots
(2 Depots – 5 vehicles required)

Figure 7. Mobility Problem Constraints and their Effect on Route Selection
infeasible solutions improves the overall quality of the search (Gendreau and Laporte 1996).

The penalty terms used in the initial testing were previously determined to be effective by Carlton (1995). Usually, the process of determining these parameters is a difficult and tedious process. Reactive penalties update the penalty parameters associated with vehicle capacity, route duration, and time windows automatically during the execution of the algorithm. All of these penalty factors are relevant to the mobility problem and a reactive search based exploring these penalties is essential to exploring the solution space of these complex problems.

Two notional MASS scenarios are presented and solved in Figures 8 and 9. The first solution to a scenario was solved in 18 seconds but is too small to display the advantages of determining routes with a heuristic approach. The larger scenario (Figure 9) provided a solution in 36 seconds and was obtained after implementing the reactive penalties in addition to previous extensions of the initial algorithm. This scenario is taken from the hub and spoke mixed integer programming model presented by Cox (1998). The scope of this multi depot problem starts to display the enhanced capabilities of a heuristic approach. We note that integer-programming approach for this scenario required 18 to 94 hours to solve using version 3.0 CPLEX solver on a Sun Sparc station 10.
Maximum Number of Vehicles: 10
-Iterations: 2500
-Capacity: 112
-Time Window Penalty Factor: 8.0
-Load Overage Penalty Factor: 10.0
-Best Feasible Cost/Travel Time Z(t): 4773.0
-Best Tour Distance (travel time - serv. time): 4663.0
-Best Feasible total wait 830.1
-Best Feasible vehicle count: 4
-Best Feasible Time of Search: 9
-Best feasible iteration: 1271

Total Time of Search: 18

---

Figure 8: SOUTHWEST ASIA SCENARIO
Maximum Number of Vehicles: 10
Iterations: 4000
Capacity: 112
Time Window Penalty Factor: 1000.0
Load Overage Penalty Factor: 1000.0
Best Feasible Cost/Travel Time Z(t): 5173.3
Best Feasible Vehicle Count: 4
Best Feasible Time of Search: 23
Best Feasible Iteration: 2655
Total Time of Search: 36

Figure 9: PACIFIC SCENARIO
2.7 Future Research

As heuristic research advances, it is more common to see two heuristic methods combined in a composite algorithm to achieve a better overall performance, as first observed by Ball and Magazine (1981). The benefits of randomness enjoyed by SA approaches can be employed in this RTS by an automatic restart capability. This extension would employ a random starting tour when a fixed number of iterations fails to find an improving solution. Based on testing, this addition appears to be more useful in solving smaller problems (25 customer). In larger problems, the best RTS solutions were obtained with a greater number of iterations; consequently, a reset feature would only be useful for searches involving large number of iterations. An approach similar to this using several constructive algorithms as starting points for solving TSP problems is presented by the Jump Search algorithm (Tsubakitani and Evans 1998). Using several good starting points and a simple local search, this algorithm shows improved results compared to a pure TS algorithm. The development of these composite approaches show promise in solving today’s applied combinatorial problems.

One important extension of this project that can be employed is the explicit consideration of non-homogeneous vehicles. Modeling nonhomogenous vehicles is straightforward since the defining attributes are capacity and airspeed. Specifically, capacity can be based on a vehicle type, while airspeed requires another time matrix to be calculated for each vehicle.

The multiple depot problems presented are intended only to show the promise of the RTS algorithm. Extending the algorithm to an approach similar to that of Renaud et al. (1996) should be accomplished to efficiently solve larger multiple depot vehicle
routing problems. In this Fast improvement Intensification and Diversification algorithm (FIND) each customer is initially assigned to its nearest depot, and then a heuristic is applied to each depot’s customer set. The fast improvement is accomplished by repeatedly applying three different types of exchanges, inter-depot (2-route exchanges between routes of different depots), intra-depot (2-route exchange between routes of the same depot) and 3-route (exchange vertices between three routes). The intensification step works on one depot at a time employing the intra-depot step to each depot in turn until no improvement is accomplished for 300 consecutive iterations. Finally, the diversification is accomplished through the repeated steps of best reinsertion between depots and inter-depot and intra depot steps while preventing moves that are tabu using a random tabulength.

2.8 Conclusion

Currently no optimization efforts are employed in MASS simulations. Earlier approaches tackled this same problem by considering only distance and route length constraints using two separate programs with run times exceeding half an hour. By contrast, our RTS algorithm can efficiently pick routes while explicitly incorporating distance, time windows, winds, vehicle capacity, vehicle range, service time, multiple depots, and -- with minor alterations -- heterogeneous vehicles. Written in the object-oriented Java programming language, it is a metaheuristic algorithm capable of running on any computer and solving large problems on a standard laptop PC in a fraction of the time required by deterministic approaches.
Previous route selection efforts outside of stochastic simulation, has centered
deterministic efforts with the k-shortest path (Rink 1997), math programming (Cox 1998)
and NRMO with a direct delivery deterministic linear programming model. Although
useful for the purpose for which they were designed, all efforts are limited by the
excessive computational time and effort required to solve complex routing problem in a
mobility scenario.

The final goal of this research effort was to provide a software application that
will provide a set of prioritized routes that will be used as a direct input into MASS. This
automated and efficient route selection tool will provide quick and near optimum route
selection without the need for an experienced analyst and numerous simulation
replications needed in a trial and error approach. Although further development and
calibration is necessary to accurately model the mobility system, many of the
characteristics and considerations that comprise this complicated system can effectively
be employed in this efficient yet powerful heuristic.

The benefits of optimizing tools are currently being realized throughout various
transportation networks from snowplows to garbage trucks and from Delta Airlines to the
United Parcel Service. In today’s world of increasing technology and shrinking route
infrastructure, the United States Air Force and Air Mobility Command can hardly afford
not to implement available, proven, and smart algorithms in its modeling and airlift
operations to increase efficiency, capability and Global Reach.
Appendix A: Extended Problem Formulation

Throughout my research I have encountered many good articles that are in turn used as reference sources for other published journals. But there is one particular reference that is used time and time again, in almost every article, journal or book written on the vehicle routing problem. This is the special issue “Routing and Scheduling of Vehicles and Crews, The State of the Art”, Computers & Operations Research written by Lawrence Bodin, Bruce Golden, Arjang Assad, and Michael Ball (1983).

This 146-page special edition makes up the entire journal issue and covers a range of related problems such as the traveling salesman problem, vehicle routing problem, crew scheduling problem and combined routing and scheduling problems. The topics included in each section include a review of the problem background, formulation, and algorithms used to solve the problem. Although the content of this article is extensive and thorough, some sections such as current heuristic approaches suffer from the fact that it was published 16 years ago. Fortunately, the underlying basic formulation is unaffected and as relevant as ever.

Bodin et al. continues the discussion of routing problems into combining crew scheduling problems and vehicle routing problems. Unfortunately, the problem I want to formulate involves an expansion of the multiple depot VRP with multiple non-homogeneous vehicles. With minor changes in notation, I am able to pick up the formulation of this problem with the aid of Carlton’s dissertation “A Tabu Search Approach to the General Vehicle Routing Problem” (1995).
A.1 Traveling Salesman Problem

The basic building block for studying the VRP is the traveling salesman problem (TSP). Without fully understanding the TSP, you can not hope to formulate and solve the more complex problem of the VRP. For this reason it is important to review the basic formulation of this problem. The first step is defining the TSP. Let $G$ be our network with the set of nodes ($N$) and a set of branches ($A$) where and the associated costs of these branches is $C = c_{ij}$. Let’s also assume that the costs are symmetric ($c_{ij} = c_{ji}$). The objective of this problem is to form a tour over all the nodes beginning and ending at the origin (node 1), which gives the minimum total tour length or cost.

The first half of this problem is the formulation of an assignment problem with only one arc ($x_{ij}$) starting at node $i$, and only one arc ($x_{ij}$) terminating at node $j$, for every node in $N$.

$$x_{ij} = \begin{cases} 
1 & \text{if arc } i-j \text{ is in the tour} \\
0 & \text{otherwise}
\end{cases}$$

Minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$

$$\sum_{i=1}^{n} x_{ij} = b_j = 1 \quad (j = 1, \ldots, n)$$

$$\sum_{j=1}^{n} x_{ij} = a_i = 1 \quad (i = 1, \ldots, n)$$

$$X = (x_{ij}) \in S \quad x_{ij} = 0 \text{ or } 1 \quad (i, j = 1, \ldots, n)$$

This is not the complete problem however, because subtours are not eliminated by this formulation. This is accomplished by the inclusion of subtour breaking constraints.
These constraints along with the assignment formulation prevent subtours from being formed. There are basically three different ways to represent the subtour breaking constraint (Bodin et al. 1983).

\[ S = \{(x_{ij}) : \sum_{i \in Q} \sum_{j \in Q} x_{ij} \geq 1 \text{ for every nonempty proper subset } Q \text{ of } N\}; \]

\[ S = \{(x_{ij}) : \sum_{i \in R} \sum_{j \in R} x_{ij} \leq |R| - 1 \text{ for every nonempty subset } R \text{ of } \{2, 3, \ldots, n\}\}; \]

\[ S = \{(x_{ij}) : y_i - y_j + nx_{ij} \leq n - 1 \text{ for } 2 \leq i \neq j \leq n \text{ for some real numbers } y_i\}. \]

The first representation states that every node subset (Q) of the set of nodes N must be connected to the other nodes in the solution. The second representation states that the arcs selected in our solution contain no cycles because if a cycle is present on R nodes, the solution must contain at least \( |R|! \) arcs. The third constrain is not so straightforward and needs a little more explanation. For this constraint let's define \( y_i \) as:

\[ y_i = \begin{cases} 
  t & \text{if node } i \text{ is visited on the } t^{th} \text{ step in a tour} \\
  0 & \text{otherwise.} 
\end{cases} \]

If an arc in the solution tour \((x_{ij} = 1)\), this constraint becomes

\[ t - (t + 1) + n \leq n - 1. \]

Conversely, everything outside the solution \((x_{ij} = 0)\) simply reduces to

\[ y_i - y_j \leq n - 1. \]

This third representation does have an advantage over the other two, adding only \( n^2 - 3n + 2 \) constraints, whereas the previous two add \( 2^n \) subtour breaking constraints to the problem's formulation (Bodin et al. 1983).
A.2 Multiple Traveling Salesman Problem

The next level of complexity in building up to the VRP is the addition of more salesman to the problem, creating the multiple traveling salesman problem (MTSP). Let $M$ be the number of salesman or vehicles that make up our fleet. Our objective, once again, is to minimize the total distance traveled. We will assume that $M$ salesman depart from the same depot and that each customer must be visited only once, and by only one salesman. Even with these changes the formulation is only an extension of the basic TSP presented earlier and is displayed below.

$$\text{Minimize } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = b_j = \begin{cases} M & \text{if } j = 1 \\ 1 & \text{if } j = 2, 3, \ldots, n \end{cases}$$

$$\sum_{j=1}^{n} x_{ij} = a_i = \begin{cases} M & \text{if } i = 1 \\ 1 & \text{if } i = 2, 3, \ldots, n \end{cases}$$

$X = (x_{ij}) \in S$

$x_{ij} = 0 \text{ or } 1 \quad (i, j = 1, \ldots, n)$

The first constraint in the formulation requires that all salesmen be used by forcing them to leave the depot. Likewise the second constraint requires all salesman to return to the depot. Any one of the subtour breaking constraints used earlier in the TSP can be used for the MTSP.

The apparent complexity of this new problem can be solved by simply reducing the MTSP to $M$ copies of the TSP. This is accomplished by creating dummy depots $(D_1, \ldots, D_M)$ all connected to the original network. These $M$ copies are either not connected, or are connected by cost prohibitive arcs. By transforming these single TSP
copies back to one common depot, the problem is now a series of $M$ subtours, which is the MTSP. This relatively straightforward transformation of the MTSP helps us understand why an algorithm used to solve a TSP, can be used to solve a MTSP (Bodin et al. 1983).

A.3 Vehicle Routing Problem

The VRP can be viewed as an extension of the TSP, obtained by adding a capacity constraint to the salesman or vehicles. The VRP involves a number of vehicles $(w)$ leaving a depot and servicing a number of customers $(n)$, each with a unique demand $(d_i)$. Each vehicle $(v)$ has a limited capacity $(K_v)$ and maximum time length for a route $(T_v)$ that constrains their closed delivery routes. This particular instance of the VRP is commonly known as the general vehicle routing problem (GVRP). If the route length or range constraints are removed, then we refer to this problem as the standard vehicle routing problem (SVRP) (Bodin et. al, 1983). In addition to the cost $(c_{ij})$ or travel time of using an arc, consider the time required to for a vehicle $v$ to deliver or service at node $i$ is $s_i^v$, travel time for vehicle $v$ from node $i$ to node $j$ as $t_{ij}^v$, and finally $x_{ij}^v = 1$ if arc $i$-$j$ is used by vehicle $v$. From this, the formulation of the GVRP follows:

$$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{w} \sum_{v=1}^{w} c_{ij}^v x_{ij}^v$$

(3.1)

$$\text{Subject to } \sum_{j=1}^{n} \sum_{v=1}^{w} x_{ij}^v = 1 \quad (j = 2, \ldots, n)$$

(3.2)

$$\sum_{j=1}^{n} \sum_{i=1}^{n} x_{ij}^v = 1 \quad (i = 2, \ldots, n)$$

(3.3)
\[
\sum_{i=1}^{n} x_{ip}^\nu - \sum_{j=1}^{n} x_{pj}^\nu = 0 \quad (v = 1, \ldots, w; \ p = 1, \ldots, n) \\
\sum_{i=1}^{n} d_i (\sum_{j=1}^{n} x_{ij}^\nu) \leq K_v \quad (v = 1, \ldots, w) \\
\sum_{i=1}^{n} s_i \sum_{j=1}^{n} x_{ij}^\nu + \sum_{i=1}^{n} \sum_{j=1}^{n} t^\prime_{ij} x_{ij}^\nu \leq T_v \quad (v = 1, \ldots, w) \\
\sum_{j=2}^{n} x_{ij}^\nu \leq 1 \quad (v = 1, \ldots, w) \\
\sum_{i=2}^{n} x_{ii}^\nu \leq 1 \quad (v = 1, \ldots, w) \\
X \in S \quad x_{ij}^\nu = 0 \text{ or } 1 \quad \text{for all } i, j, v
\]

The objective function (3.1), minimizing the overall distance, remains the same but is formulated by summing over all the vehicles. Equations (3.2) and (3.3) make sure every customer is visited by one and only one vehicle. It is important to note that we are assuming that a customer’s demand does not exceed vehicle capacity and each customer is fully serviced by the one vehicle that visits it. Equation (3.4) checks the continuity of our routes while (3.5) maintains the capacity constraint on all of the vehicles. Since we are representing route length restrictions by time, we use Equation (3.6) to insure maximum route time is not exceeded. Finally, Equations (3.7) and (3.8) insure that we do not use more vehicles than we have.

In addition to these equations we must once again include our subtour breaking constraints that will entail a slight modification to those used earlier in the TSP. Since the third subtour representation is the most efficient, we will use that formulation and expand it.
\[ S = \{ (x_{ij}^v) : \ y_i^v - y_j^v + nx_{ij}^v \leq n - 1 \ \text{for} \ 2 \leq i \neq j \leq n \ \text{for some real numbers} \ y_i^v \} \]

This simply applies the original subtour breaking constraint to each vehicle in turn. We can also eliminate some redundant constraints from the formulation above. Using (3.2) and (3.4) enforces (3.3) automatically and makes it unnecessary (Bodin et al. 1983). Likewise (3.4) and (3.7) imply (3.8) so this too can be eliminated from the formulation (Bodin et al. 1983).

Finally, one common restriction added to the VRP is time windows. Let \( a_j \) be the arrival time to node \( j \), \( e_j \) be the earliest delivery time allowable and \( l_j \) be the no later than time for delivery. Using a nonlinear representation we get:

\[
a_j = \sum_{v=1}^{w} \sum_{i=1}^{n} (a_i + s_i^v + t_{ij}^v)x_{ij}^v \quad (j = 1, \ldots, n)
\]

\[
a_f = 0
\]

\[
e_j \leq a_j \leq l_j \quad (j = 2, \ldots, n)
\]

For each \( j \), one of the \( x_{ij}^v \) variables equals 1, so \( a_j \) is the sum of the previous arrival time (\( a_i \)), the service time at node \( i \) (\( s_i^v \)), and the travel time from \( i \) to \( j \) (\( t_{ij}^v \)). Alternatively we can use the linear representation of time windows constraint in the formulation (Bodin et al. 1983).

\[
a_j \geq (a_i + s_i^v + t_{ij}^v) - (1 - x_{ij}^v) T_{\text{max}}^v
\]

\[
a_j \leq (a_i + s_i^v + t_{ij}^v) + (1 - x_{ij}^v) T_{\text{max}}^v
\]

for all \( i, j, v \)

When \( x_{ij}^v = 1 \), the second half of the equation is eliminated and \( a_j \) is simply determined from the previous arrival time, previous service time and the travel time between the nodes. On the other hand, when \( x_{ij}^v = 0 \), the constraints are redundant.
A.4 Multiple Depot VRP

Expanding the previous GVRP to account for multiple bases of operation or depots gives us the multiple depot VRP. This problem can be formulated with only minor changes. Let $M$ be the number of depots in our problem. First the original VRP formulation indexes are changed for equation (3.2), $(i = M+1,\ldots,n)$, and equation (3.3), $(i = M+1,\ldots,n)$. Next the constraints (3.7) and (3.8) must be changed to sum over all the depots individually in order to check that the number of vehicles being used does not exceed the number of vehicles on hand.

$$\sum_{i=1}^{M} \sum_{j=M+1}^{n} x_{ij}^{v} \leq 1 \quad (v = 1,\ldots,w)$$

$$\sum_{j=1}^{n} \sum_{i=M+1}^{n} x_{ij}^{v} \leq 1 \quad (v = 1,\ldots,w)$$

Of course, this change also includes an adjustment of the subtour breaking constraint. Although only one is used, we will show the changes for all three (Bodin et al. 1983).

$$S = \{(x_{ij}): \sum_{i \in Q} \sum_{j \not\in Q} x_{ij} \geq 1 \text{ for every non empty proper subset } Q \text{ of } \{1,2,\ldots,n\} \text{ containing nodes } 1, 2, \ldots, M\};$$

$$S = \{(x_{ij}): \sum_{i \in R} \sum_{j \in R} x_{ij} \leq |R|-1 \text{ for every nonempty subset } R \text{ of } \{M+1,M+2,\ldots,n\}\};$$

$$S = \{(x_{ij}): y_{i} - y_{j} + n x_{ij} \leq n - 1 \text{ for } M + 1 \leq i \neq j \leq n \text{ for some real numbers } y_{i}\}.$$

At this point the article, Bodin et al. continues into the discussion of combining crew scheduling problems and vehicle routing problems. Unfortunately, the problem I want to formulate involves an expansion of the multiple depot VRP to a multiple non-
homogeneous vehicle pick up and delivery problem. With minor changes in notation, I am able to pick up the formulation of this problem with the aid of Carlton (1995).

A.5 Pickup and Delivery Problem

The pickup and delivery problem (PDP) is a VRP that adds the precedence constraint. Precedence means that a package must be picked up at node $i$ before it can be delivered to node $j$. With this added constraint, and some minor changes in notation, we finally arrive at the one of the most general routing problems studied. Simpler problems that must be formulated are simply a relaxation of this problem.

In this formulation, a superscript of $(v, r)$ will be used corresponding to the specific vehicle $v$ assigned to depot $r$. The customers are still indexed by $i$ or $j$, each requiring a load $d_i$, to be picked up and delivered from node $i$ to location $n + i$. The set of all depots is defined as $D$ and the set of all vehicles as $V$.

The set of pickup locations are $P^+$, where $|P^+| = n$, and the pickup locations will be numbered from 1 to $n$. The set of delivery locations are $P^-$, where $|P^-| = n$, and these delivery locations will be numbered from $n + 1$ to $2n$. The set of all pickup and delivery locations, $(P^+ \cup P^-)$, will be $P$ and the set of all modeled pickup and delivery locations, customers and depots, will be referred to as $N$. Customer subscripts referring to a depot at the beginning of a tour are annotated as 0 and those at the end of a tour are labeled $2n+1$.

We also introduce a load variable $Y_i$, indicating the total vehicle load at customer $i$. With these changes, the formulation of the multiple depot, multiple non-homogeneous vehicle, route length constrained, PDP with time windows is:
Objective Function

\[
\text{Minimize} \sum_{r \in D} \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} c_{ij}^{vr} x_{ij}^{vr}
\]

Subject to:

Tour constraints:

\[
\sum_{r \in D} \sum_{v \in V} \sum_{j \in N} x_{ij}^{vr} = 1 \quad \forall \ i \in P^+
\]

\[
\sum_{j \in N} x_{ij}^{vr} - \sum_{j \in N} x_{ij}^{vr} = 0 \quad \forall \ i \in P, \ v \in V, \ r \in D
\]

\[
\sum_{j \in P^*} x_{i,0,j}^{vr} = 1 \quad \forall \ v \in V, \ r \in D
\]

\[
\sum_{j \in P^*} x_{i,2n+1}^{vr} = 1 \quad \forall \ v \in V, \ r \in D
\]

\[
\sum_{j \in N} x_{ij}^{vr} - \sum_{j \in N} x_{j,i+1}^{vr} = 0 \quad \forall \ i \in P^+, \ v \in V, \ r \in D
\]

Precedence constraints:

\[
a_i + s_i + t_{i,n+1}^{vr} \leq a_{n+1} \quad \forall \ i \in P^+
\]

If \( x_{ij}^{vr} = 1 \) then:

\[
a_i + s_i + t_{i,n+1}^{vr} \leq a_j \quad \forall \ i \in P, \ v \in V, \ r \in D
\]

If \( x_{0,j}^{vr} = 1 \) then:

\[
a_0^{vr} + t_{0,j}^{vr} \leq a_j \quad \forall \ i \in P^+, \ v \in V, \ r \in D
\]

If \( x_{i,2n+1}^{vr} = 1 \) then:

\[
a_i + s_i + t_{i,2n+1}^{vr} \leq a_{2n+1}^{vr} \quad \forall \ i \in P^+, \ v \in V, \ r \in D
\]

Capacity constraints:

If \( x_{ij}^{vr} = 1 \) then:

\[
y_i^{vr} + d_j = y_j^{vr} \quad \forall \ i \in P, \ j \in P^+, \ v \in V, \ r \in D
\]

If \( x_{ij}^{vr} = 1 \) then:

\[
y_i^{vr} + d_{j-n} = y_j^{vr} \quad \forall \ i \in P, \ j \in P^+, \ v \in V, \ r \in D
\]

If \( x_{0,j}^{vr} = 1 \) then:

\[
y_0^{vr} + d_j = y_j^{vr} \quad \forall \ j \in P^+, \ v \in V, \ r \in D
\]
\[ Y_{0v}^r = 0 \quad \forall v \in V, \, r \in D \]  (4.13)

\[ 0 \leq Y_{iv}^r \leq K_{iv}^r \quad \forall \, i \in P^+, \, v \in V, \, r \in D \]  (4.14)

**Time Window Constraints:**

\[ e_i \leq a_i \leq l_i \quad \forall \, i \in P \]  (4.15)

\[ e_0^r \leq a_0^r \leq l_0^r \quad \forall \, v \in V, \, r \in D \]  (4.16)

\[ e_{2n+1}^r \leq a_{2n+1}^r \leq l_{2n+1}^r \quad \forall \, v \in V, \, r \in D \]  (4.17)

**Binary Constraints:**

\[ x_{ij}^r \in \{0,1\} \quad \forall \, i,j \in N, \, v \in V, \, r \in D \]

With the exception of the expanded notation, many of the constraints remain the same as those presented in earlier problems. The first group of constraints (4.1) - (4.5) are responsible for building the tours. Constraints (4.3) and (4.4) are responsible for making sure all vehicles are used by making them leave and return to the depot. If it is not necessary to use all vehicles in the problem then we can change the equality to a less than or equal to sign (Carlton, 1995). Finally constraint (4.5) requires the same vehicle that picks up a package to deliver it.

The precedence constraints (4.6) - (4.9) are the next group of constraints. When presented this way, the subtour breaking constraint used before, is essentially included in this formulation (Carlton, 1995). The use of service time and travel time insures a time order sequence of routes. The capacity constraints (4.10) - (4.14) are now tracked at every node as well as by vehicle. Finally, the representation of time windows (4.15) -
(4.17) is expanded to include hard time windows leaving and returning to the depots which enforces a limit on the possible route length.
Appendix B: Java Documentation

Class Hierarchy
- class java.lang.Object
- class `Convert`
- class `CoordType`
- class `CycleOut`
- class `HashMod`
- class `InFromKeybd`
- class `KeyObj`
- class `KeyToString`
- class `MTSPTW`
- class `BestSolnMod`
- class `TsptwPen`
- class `NoCycleOut`
- class `NodeType`
- class `PrintCalls`
- class `PrintFlag`
- class `ReacTabuObj`
- class `ReadFile`
- class `SearchOut`
- class `StartPenBestOut`
- class `StartTourObj`
- class `TabuMod`
- class `TimeMatrixObj`
- class `Timer`
- class `TsptwPenOut`
- class `TwBestTTOut`
- class `ValueObj`
- class `VrpPenType`

Index of all Fields and Methods

A

`assignInputFile(String)`. Static method in class `ReadFile`
`assignInputFile` sets up the FileInputStream.

B

`bearingXY(CoordType, CoordType, double)`. Static method in class `Convert`
`bearingXY` calculates the true bearing (in degrees) from one coordinate point to the second coordinate point and returns the value as a double precision number.

`bestCost`. Variable in class `SearchOut`
`bestCost`. Variable in class `StartPenBestOut`
Penalty related value.

**bestCost**. Variable in class **TwBestTTOut**
  best tour related value.

**bestiter**. Variable in class **SearchOut**
**bestiter**. Variable in class **StartPenBestOut**
  Penalty related value.
**bestiter**. Variable in class **TwBestTTOut**
  best tour related value.

**bestny**. Variable in class **SearchOut**
**bestny**. Variable in class **StartPenBestOut**
  Penalty related value.
**bestny**. Variable in class **TwBestTTOut**
  best tour related value.

**BestSolnMod()**. Constructor for class **BestSolnMod**
**bestTime**. Variable in class **SearchOut**
**bestTime**. Variable in class **StartPenBestOut**
  Penalty related value.
**bestTime**. Variable in class **TwBestTTOut**
  best tour related value.

**bestTour**. Variable in class **SearchOut**
**bestTour**. Variable in class **StartPenBestOut**
  Saved tour.
**bestTour**. Variable in class **TwBestTTOut**
  best tour related value.

**bestTT**. Variable in class **SearchOut**
**bestTT**. Variable in class **StartPenBestOut**
  Penalty related value.
**bestTT**. Variable in class **TwBestTTOut**
  best tour related value.

**bfCost**. Variable in class **SearchOut**
**bfCost**. Variable in class **StartPenBestOut**
  Penalty related value.
**bfCost**. Variable in class **TwBestTTOut**
  best tour related value.

**bfiter**. Variable in class **SearchOut**
**bfiter**. Variable in class **StartPenBestOut**
  Penalty related value.
**bfiter**. Variable in class **TwBestTTOut**
  best tour related value.

**bfny**. Variable in class **SearchOut**
**bfny**. Variable in class **StartPenBestOut**
  Penalty related value.
**bfny**. Variable in class **TwBestTTOut**
  best tour related value.

**bfTime**. Variable in class **SearchOut**
**bfTime**. Variable in class **StartPenBestOut**
  Penalty related value.
**bfTime**. Variable in class **TwBestTTOut**
  best tour related value.

**bfTour**. Variable in class **SearchOut**
**bfTour**. Variable in class **StartPenBestOut**
  Saved tour.
**bfTour**. Variable in class **TwBestTTOut**
  best tour related value.

**bfTT**. Variable in class **SearchOut**
**bfTT**. Variable in class **StartPenBestOut**
Penalty related value.

**bFT** Variable in class TwBestTTOut
best tour related value.

C

**compPens**(NodeType[], int). Static method in class NodeType
compPens computes the vehicle capacity overload and time window penalties.

**compPens**(NodeType[], int). Method in class VrpPenType
compPens computes the vehicle capacity overload and time window penalties.

**Convert()**. Constructor for class Convert

**CoordType()**. Constructor for class CoordType
Default constructor.

**CoordType**(String, double, double). Constructor for class CoordType
Lat/long constructor.

**copy()**. Method in class NodeType

**countVeh**(NodeType[]). Static method in class NodeType
Method countVeh finds the number of vehicles being used in the current tour by counting the
vehicle to demand transitions.

**countVehicles**(NodeType[]). Static method in class TabuMod
countVeh method calculates the number of vehicles used in the current tour by counting the
number of vehicle (type 2) to demand (type 1) transitions.

**cycle**(ValueObj, double, int, int, int, double, int, int, PrintFlag). Static method in class TabuMod
cycle method updates the search parameters if the incumbent tour is found in the hashing structure.

**CycleOut()**. Constructor for class CycleOut
Default constructor.

**CycleOut**(int, int, double, ValueObj). Constructor for class CycleOut
Specified constructor.

**cyclePrint**. Variable in class PrintFlag
print flag.

D

**distanceXY**(CoordType, CoordType). Static method in class Convert
distanceXY calculates the great circle distance (in nautical miles) between two coordinate points
and returns the value as a double precision number.

**DMMmtoDd**(int, double). Static method in class Convert
DMMmtoDd converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to
"Degrees Decimal Degrees" (D.d) format.

**DMMmtoDd**(int, double, String). Static method in class Convert
DMMmtoDd converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to
"Degrees Decimal Degrees" (D.d) format.

**DMMSSstoSd**(int, int, double). Static method in class Convert
DMMSSstoSd converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs)
format to "Degrees Decimal Degrees" (D.d) format.

**DMMSSstoSd**(int, int, double, String). Static method in class Convert
DMMSSstoSd converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs)
format to "Degrees Decimal Degrees" (D.d) format.

E

**endTime**. Variable in class Timer
end time.

**endTime()**. Method in class Timer
endTime assigns end time.

**equals**(KeyObj). Method in class KeyObj
Overloaded equals(), check only attribute fields.

**equals**(ValueObj). Method in class ValueObj
Overloaded equals(), check only attribute fields.
firstHashVal(int). Static method in class HashMod
firstHashVal method assigns the primary hashing value.

getEa(). Method in class NodeType
getId(). Method in class NodeType
getLa(). Method in class NodeType
getLoad(). Method in class NodeType
getM(). Method in class NodeType
getQty(). Method in class NodeType
getType(). Method in class NodeType
getWait(). Method in class NodeType
groundSpeed(double, double, double, double, double). Static method in class WindAdjust
groundSpeed method returns the ground speed given the heading between points, the wind
heading, the wind speed, and the aircraft's airspeed.

hashCode(). Method in class KeyObj
hashCode(). Method in class ValueObj
Overloaded hashCode method.

H
HashMod(). Constructor for class HashMod
HHMMtoMM(int). Static method in class Convert
HHMMtoMM converts a military time to the equivalent number of minutes (i.e., 0630 hours to
390 minutes) for use in time window and service time calculations.

HMMtoHh(int). Static method in class Convert
HMMtoHh converts a military specified time to the equivalent decimal hour equivalent (i.e., 0630
hours to 6.5 hours) for use in time window and service time calculations.

I
InFromKeybd(). Constructor for class InFromKeybd
insert(NodeType[], int, int). Static method in class NodeType
Method insert allows the element designated by "chI" to be shifted by "chD" elements.

iterPrint. Variable in class PrintFlag
print flag.

K
keyDouble(String). Static method in class InFromKeybd
keyDouble allows user to enter a double from the keyboard.
keyFloat(String). Static method in class InFromKeybd
keyFloat allows user to enter a float from the keyboard.
keyInt(String). Static method in class InFromKeybd
keyInt allows user to enter an integer from the keyboard.
KeyObj(int, int, int, int, int). Constructor for class KeyObj
Specified constructor.
keyString(String). Static method in class InFromKeybd
keyString allows user to enter a string from the keyboard.
KeyToString(). Constructor for class KeyToString
keyToString(int, int, int, int, int). Static method in class KeyToString
KeyToString Class converts the attributes of tour to a concatenated string used as a key to the
hashable of tours.

L
loadPrint. Variable in class PrintFlag
print flag.
lookFor(Hashtable, int, int, int, int, int, int, int). Static method in class HashMod
    lookFor method searches for the current tour in the hashing structure, if the tour is found a true
    value for the boolean "found" is returned, if not found, the tour is added to the hashtable.

M
main(String[]). Static method in class MTSPTW
    main executes MTSPTW problem.
mayg. Variable in class CycleOut
    moving average.
MMtoHMM(int). Static method in class Convert
    MMtoHMM converts a given number of minutes to a military time hour format (i.e., 390
    minutes to 0630 hours) for human friendly output.
movePrint. Variable in class PrintFlag
    print flag.
moveValTT(int, int, NodeType[], NodeType[], int[][]). Static method in class NodeType
    Method moveValTT computes the incremental change in the value of the travel time from the
    incumbent tour to the proposed neighbor tour, and computes the neighbor schedule parameters
    preparing for computation of penalty terms.
moveValTT(int, int, NodeType[], NodeType[], int[][]). Static method in class TabuMod
    Method moveValTT computes the incremental change in the value of the travel time from the
    incumbent tour to the proposed neighbor tour, and computes the neighbor schedule parameters
    preparing for computation of penalty terms.
MTSPTW(). Constructor for class MTSPTW
N
noCycle(double, int, double, int, int, PrintFlag). Static method in class TabuMod
    noCycle method updates the search parameters if the incumbent tour is not found in the hashing
    structure.
NoCycleOut(). Constructor for class NoCycleOut
    Default constructor.
NoCycleOut(int, int). Constructor for class NoCycleOut
    Specified constructor.
NodeType(). Constructor for class NodeType
    Default constructor.
NodeType(int, int, int, int, int, int). Constructor for class NodeType
    Specified constructor.
umfeas. Variable in class SearchOut
P
penTrav. Variable in class SearchOut
penTrav. Variable in class StartPenBestOut
    Penalty related value.
penTrav. Variable in class TsptwPenOut
    Penalty related value.
print(). Method in class NodeType
PrintCalls(). Constructor for class PrintCalls
PrintFlag(). Constructor for class PrintFlag
    Default PrintFlag constructor sets all to "true".
PrintFlag(boolean). Constructor for class PrintFlag
    Additional PrintFlag constructor allows specification of either "true" or "false".
printInitVals(int, int, int, double, String). Static method in class PrintCalls
printTour(NodeType[]). Static method in class NodeType
R
randWtWZ(int, int, int). Static method in class HashMod
    randWtWZ method computes Woodruff & Zemel random weights between 1 & range for all
nodes.

**ReacTabuObj()**. Constructor for class ReacTabuObj

**ReadFile().** Constructor for class ReadFile

**readNC(String).** Static method in class TimeMatrixObj

readNC is used to read from the first token from the input file (the number of customers (nc)).

**readNextDouble(StreamTokenizer).** Static method in class ReadFile

readNextString method gets the next token and returns it as a double.

**readNextInt(StreamTokenizer).** Static method in class ReadFile

readNextString method gets the next token and returns it as an integer.

**readNextString(StreamTokenizer).** Static method in class ReadFile

readNextString method gets the next token and returns it as a string.

**readNV(String).** Static method in class TimeMatrixObj

readNV is used to read from the second token from the input file (the number of vehicles (nv)).

**readTSPTW(double, int, int, String, CoordType[], int[]).** Static method in class TimeMatrixObj

readTSPTW reads in the geographical coordinates and time window file and calculates the time between each node

**readTSPTWdepot(double, int, int, String, CoordType[], int[]).** Static method in class TimeMatrixObj

readTSPTWdepot reads in the geographical coordinates, load quantity, service time, and time window information associated with depot and customer locations from the input file.

**rTestStepPrint(int, int, int, int, int, int, int).** Static method in class PrintCalls

S

**search(double, double, double, double, int, int, int, int, int, int, int, VrpPenType, int[], PrintFlag, int, int, int, int, int, int, int, int, int, int, int, int, int, int, NodeStkType[], NodeStkType[], NodeStkType[]).** Static method in class ReacTabuObj

ReacTabuObj steps through iterations of the reactive tabu search.

**SearchOut().** Constructor for class SearchOut

Default constructor.

**SearchOut(int, int, int, int, int, int, int, int, int, int, int, int, int, VrpPenType, NodeStkType[], NodeStkType[], NodeStkType[]).** Constructor for class SearchOut

Specified constructor.

**secondHashVal(int, int, NodeStkType[], int[]).** Static method in class HashMod

secondHashVal updates the Woodruff & Zemel second hashing value based on the tour insertion move.

**setId(int).** Method in class NodeType

**setLoad(int).** Method in class NodeType

**setQty(int).** Method in class NodeType

**setType(int).** Method in class NodeType

**setWait(int).** Method in class NodeType

**sslC.** Variable in class CycleOut

**sslC.** Variable in class NoCycleOut

cycle related variable.

**startPenBest(int, int, NodeStkType[], double, double, double, int, int, int, int, int, int, int, VrpPenType, int, int, int, int, int, int, int, int, int, int, NodeStkType[], NodeStkType[]).** Static method in class StartTourObj

startPenBest initializes "best" values and their times.

**StartPenBestOut().** Constructor for class StartPenBestOut

Default constructor.

**StartPenBestOut(int, int, int, int, int, int, int, int, int, int, int, int, int, int, VrpPenType, NodeStkType[], NodeStkType[]).** Constructor for class StartPenBestOut

Specified constructor.

**startPrint.** Variable in class PrintFlag

print flag.

**startTime.** Variable in class Timer

begin time.

**startTime().** Method in class Timer

startTime assigns start time.
startTour(NodeType[], int[][], int, int). Static method in class NodeType
    Method startTour will bubble sort the initial tour based on the average time window time.
StartTourObj(). Constructor for class StartTourObj
stepLoopPrint. Variable in class PrintFlag
    print flag.
stepPrint. Variable in class PrintFlag
    print flag.
sumWait(NodeType[]). Static method in class NodeType
    Method sumWait calculates the total "waiting" time in a particular tour by summing the wait
values for each individual node.
swap(int, int). Static method in class MTSPTW
    Swap allows generic swap of integers.
swapInt(int, int). Static method in class NodeType
    Method swapInt switches two integers
swapNode(NodeType[], int, int). Static method in class NodeType
    Method swapNode allows the node array elements "a" and "b" to be swapped in the Node Array
    "z".

T

tabuLen. Variable in class CycleOut
tabuLen. Variable in class NoCycleOut
cycle related variable.
TabuMod(). Constructor for class TabuMod
timeMatrix(int, int, double, int, CoordType[], int[]). Static method in class TimeMatrixObj
timeMatrix computes simple two-dimensional time/distance matrix.
timeMatrixDepot(int, int, double, int, CoordType[], int[]). Static method in class TimeMatrixObj
timeMatrixDepot computes the two-dimensional array used as the "time" matrix.
TimeMatrixObj(). Constructor for class TimeMatrixObj
timePrint. Variable in class PrintFlag
    print flag.
Timer(). Constructor for class Timer
    Default constructor.
toString(). Method in class KeyObj
toString changes a KeyObj to a string for use in the hashTable.
toString(). Method in class ValueObj
toString changes a ValueObj to a string for use in the hashTable.
totalSeconds. Variable in class Timer
duration of run.
totalSeconds(). Method in class Timer
totalSeconds returns duration.
totPenalty. Variable in class SearchOut
totPenalty. Variable in class StartPenBestOut
    Penalty related value.
totPenalty. Variable in class TspwPenOut
    Penalty related value.
tour. Variable in class SearchOut
tourCost. Variable in class SearchOut
tourCost. Variable in class StartPenBestOut
    Penalty related value.
tourCost. Variable in class TspwPenOut
    Penalty related value.
tourHVwz(NodeType[], int[]). Static method in class HashMod
tourHVwz method computes the Woodruff & Zemel hashing value from the sum of adjacent node
id multiplication.
tourPen. Variable in class SearchOut
tourPen. Variable in class StartPenBestOut
Tour penalty values.

**tourSched***(int, NodeType[], int[])[]). Static method in class **NodeType**
   Method tourSched should be called with the syntax tourLen = tourSched(nodeArray, time) from
   the orderStartingTour method.

**TsptwPen**(). Constructor for class **TsptwPen**

TsptwPen*(int, NodeType[], VrpPenType, double, double, int, int, int). Static method in class **TsptwPen**
   tsptwPen method uses the TW and load penalties to computes tourCost of tour as tour length +
   scaled penalty for infeasibilities.

**tsptwPenNormalized***(int, NodeType[], VrpPenType, double, double, int, int, int, int). Static method in
   class **TsptwPen**
   tsptwPenNormalized method uses the TW and load penalties to computes tourCost of tour as tour
   length + scaled penalty for infeasibilities.

TsptwPenOut(). Constructor for class **TsptwPenOut**
   Default constructor.

TsptwPenOut*(int, int, int, int). Constructor for class **TsptwPenOut**
   Specified constructor.

*tvl*. Variable in class **SearchOut**

tvl. Variable in class **TsptwPenOut**
   Penalty related value.

twBestTT*(int, int, int, int, NodeType[], int, int, int, int, int, int, NodeTrype[], NodeTrype[],
   int, int, int, NodeTrype[], NodeTrype[], int, int). Static method in class **BestSolnMod**
   twBestTT compares current tour with previous best and best feasible tours and updates records
   accordingly.

TwBestTTOut(). Constructor for class **TwBestTTOut**
   Default constructor.

TwBestTTOut*(int, int, int, int, int, int, int, int, int, int, int, int, int, NodeTrype[], NodeTrype[]). Constructor for class
   TwBestTTOut
   Specified constructor.

twrdPrint. Variable in class **PrintFlag**
   print flag.

V

**ValueObj***(int, int, int, int, int, int). Constructor for class **ValueObj**
   Specified constructor.

VrpPenType(). Constructor for class **VrpPenType**
   Default constructor.

VrpPenType*(int, int). Constructor for class **VrpPenType**
   Specified constructor.

W

WindAdjust(). Constructor for class **WindAdjust**
Class BestSolnMod

java.lang.Object
    +-----MTSPTW
    |     +-----BestSolnMod

public class BestSolnMod extends MTSPTW BestSolnMod class retains the tours with the best travel times and tour costs.
Version:  
v1.1 Mar 99
Author:  
    Kevin P. O'Rourke, David M. Ryer

Constructor Index

BestSolnMod()

Method Index

twBestTT(int, int, int, int, int, NodeT [], int, int, int, int, int, int, NodeT [], NodeT [],  
        int, int)  
twBestTT compares current tour with previous best and best feasible tours and updates records accordingly.

Constructors

BestSolnMod

    public BestSolnMod()

Methods

twBestTT

    public static TwBestTTOut twBestTT(int numnodes,  
        int totPenalty,  
        int penTrav,  
        int tvl,  
        int ntvu,  
        int iter,  
        NodeT tour[],  
        int bfcost,  
        int bftt,  
        int bfnv,  
        int bfiter,  
        int bestCost,  
        int bestTT,  
        int bestnv,  
        int bestiter,  
        NodeT bfTour[],  
        NodeT bestTour[],  
        int bfTime,  
        int bestTime)
twBestTT compares current tour with previous best and best feasible tours and updates records accordingly.

Returns:
returns packages output object.

**Class Convert**

```java
public class Convert
extends Object
Convert contains general conversion formulas applicable to location and distance calculations. Included are conversions between decimal format and hours-minutes-seconds format, great circle distance between two specified coordinates, and bearing from one point to another.
```

Version:
v1.1 Feb 99

Author:
Kevin P. O'Rourke, David M. Ryer

**Constructor Index**

**Convert()**

**Method Index**

- `bearingXY(CoordType, CoordType, double)`
  - `bearingXY` calculates the true bearing (in degrees) from one coordinate point to the second coordinate point and returns the value as a double precision number.

- `distanceXY(CoordType, CoordType)`
  - `distanceXY` calculates the great circle distance (in nautical miles) between two coordinate points and returns the value as a double precision number.

- `DMMntoDd(int, double)`
  - `DMMntoDd` converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to "Degrees Decimal Degrees" (D.d) format.

- `DMMntoDd(int, double, String)`
  - `DMMntoDd` converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to "Degrees Decimal Degrees" (D.d) format.

- `DMMSSstoDd(int, int, double)`
  - `DMMSSstoDd` converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs) format to "Degrees Decimal Degrees" (D.d) format.

- `DMMSSstoDd(int, int, double, String)`
  - `DMMSSstoDd` converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs) format to "Degrees Decimal Degrees" (D.d) format.

- `HHMMtoMM(int)`
  - `HHMMtoMM` converts a military time to the equivalent number of minutes (i.e., 0630 hours to 390 minutes) for use in time window and service time calculations.

- `HHMntoHh(int)`
  - `HHMntoHh` converts a military specified time to the equivalent decimal hour equivalent (i.e., 0630 hours to 6.5 hours) for use in time window and service time calculations.

- `MMstoHHMM(int)`
  - `MMstoHHMM` converts a given number of minutes to a military time hour format (i.e., 390 minutes to 0630 hours) for human friendly output.

58
Constructors

Convert

public Convert()

Methods

DMMmtoDd

public static double DMMmtoDd(int degrees, double minutes)

DMMmtoDd converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to "Degrees Decimal Degrees" (D.d) format. The D.MMm is the "human friendly" form of the data. The D.d format is required to readily perform distance calculations.

Parameters:

degrees - integer degree value of coordinate.
minutes - double minute value of coordinate.

Returns:

returns double Dd coordinate in the "degrees decimal degrees" format.

DMMmtoDd

public static double DMMmtoDd(int degrees, double minutes, String name)

DMMmtoDd converts a number in "Degrees Minutes Decimal Minutes" (D.MMm) format to "Degrees Decimal Degrees" (D.d) format. The D.MMm is the "human friendly" form of the data. The D.d format is required to readily perform distance calculations. This version of the method considers hemisphere and assigns a negative value if appropriate to south and east coordinates.

Parameters:

degrees - integer degree value of coordinate.
minutes - double minute value of coordinate.
name - string hemisphere value of coordinate (either "E", "W", "N", or "S").

Returns:

returns Dd coordinate in the "degrees decimal degrees" format.

DMMSsstoDd

public static double DMMSsstoDd(int degrees, int minutes, double seconds)

DMMSsstoDd converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs) format to "Degrees Decimal Degrees" (D.d) format. The D.MMSSs is the "human friendly" form of the data. The D.d format is required to readily perform distance calculations.

Parameters:

degrees - integer degree value of coordinate.
minutes - integer minute value of coordinate.
seconds - double second value of coordinate.

Returns:

returns Dd coordinate in the "degrees decimal degrees" format.

DMMSsstoDd

public static double DMMSsstoDd(int degrees, int minutes, double seconds, String name)

DMMSsstoDd converts a number in "Degrees Minutes Seconds Decimal Seconds" (D.MMSSs) format to "Degrees Decimal Degrees" (D.d) format. The D.MMSSs is the "human friendly" form
of the data. The D.d format is required to readily perform distance calculations. This version of the method considers hemisphere and assigns a negative value if appropriate to south and east coordinates.

Parameters:
degrees - integer degree value of coordinate.
minutes - integer minute value of coordinate.
seconds - double second value of coordinate.
name - string hemisphere value of coordinate (either "E", "W", "N", or "S").

Returns:
returns Dd coordinate in the "degrees decimal degrees" format.

HMMtoHh

public static double HMMtoHh(int time)
HMMtoHh converts a military specified time to the equivalent decimal hour equivalent (i.e., 0630 hours to 6.5 hours) for use in time window and service time calculations.

Parameters:
time - integer whole minute "military format" (0630 hours) time value.

Returns:
returns Hh double fractional hour (6.5 hours) time value.

HHMMtoMM

public static int HHMMtoMM(int time)
HHMMtoMM converts a military time to the equivalent number of minutes (i.e., 0630 hours to 390 minutes) for use in time window and service time calculations.

Parameters:
time - integer whole minute "military format" (0630 hours) time value.

Returns:
returns MM integer number of minutes (390 minutes) time value.

MMtoHHMM

public static int MMtoHHMM(int time)
MMtoHHMM converts a given number of minutes to a military time hour format (i.e., 390 minutes to 0630 hours) for human friendly output.

Parameters:
time - integer number of minutes (390 minutes) time value.

Returns:
returns HHMM integer whole minute "military format" (0630 hours) time value.

distanceXY

public static double distanceXY(CoordType x,
                                 CoordType y)
distanceXY calculates the great circle distance (in nautical miles) between two coordinate points and returns the value as a double precision number.

Parameters:
x - CoordType coordinate of first position.
y - CoordType coordinate of second position.

Returns:
returns distanceXY double distance between the two points in nautical miles.

bearingXY

public static double bearingXY(CoordType x,
                                 CoordType y,
                                 double dXY)
bearingXY calculates the true bearing (in degrees) from one coordinate point to the second coordinate point and returns the value as a double precision number.
Parameters:
  x - CoordType coordinate of first position.
  y - CoordType coordinate of second position.
  dXY - double distance between the first and second position, in nautical miles.
Returns:
  returns thetaXY double initial true heading from the first point to the second point measured from true north in degrees.

Class CoordType

java.lang.Object
   |   +----CoordType

public class CoordType extends Object CoordType is used to hold coordinate location for customer/vehicle nodes. It contains fields for both x, y integer data and lat/long data, although only one set will be used.
Version:
   v1.1 Feb 99
Author:
   Kevin P. O'Rourke, David M. Ryer

Constructor Index

CoordType()
   Default constructor.
CoordType(String, double, double)
   Lat/long constructor.

Constructors

CoordType

   public CoordType()
      Default constructor. Assigns name to null and all values to zero.

CoordType

   public CoordType(String nameLabel,
                    double lat,
                    double lon)
      Lat/long constructor. Assigns name, latitude, and longitude as specified.

Class CycleOut

java.lang.Object
   |   +----CycleOut

public class CycleOut extends Object CycleOut is used as a package to output multiple fields from the class Cycle.
Version:
   v1.1 Mar 99
Author:
Variable Index

\textbf{mavg}
  \textit{moving average.}

\textbf{ssltc}

\textbf{tabuLen}

Constructor Index

\textbf{CycleOut()}
  \textit{Default constructor.}

\textbf{CycleOut(int, int, double, ValueObj)}
  \textit{Specified constructor.}

\textbf{ssltc}

\texttt{public int ssltc}

\textbf{tabuLen}

\texttt{public int tabuLen}

\textbf{mavg}

\texttt{public double mavg}
  \textit{moving average.}

Constructors

\textbf{CycleOut}

\texttt{public CycleOut()}
  \textit{Default constructor. Assigns all values to zero.}

\textbf{CycleOut}

\texttt{public CycleOut(int ssltc,}
  \texttt{  int tabuLen,}
  \texttt{  double mavg,}
  \texttt{  ValueObj matchPtr)}
  \textit{Specified constructor. Values set as passed.}

Class HashMod

\texttt{java.lang.Object}
  \texttt{|}
  \texttt{-----HashMod}

\texttt{public class HashMod}
\texttt{extends Object} HashMod Class contains methods to assign first and second hashing values (used in the hashtable) and the search method to search the hashtable.

Version:
  \texttt{v1.1 Mar 99}

Author:
  \texttt{Kevin P. O'Rourke, David M. Ryer}
Constructor Index

HashMod()

Method Index

firstHashVal(int)
    firstHashVal method assigns the primary hashing value.

lookFor(Hashtable, int, int, int, int, int, int, int)
    lookFor method searches for the current tour in the hashing structure, if the tour is found a true value for the boolean "found" is returned, if not found, the tour is added to the hashtable.

randWtWZ(int, int, int)
    randWtWZ method computes Woodruff & Zemel random weights between 1 & range for all nodes.

secondHashVal(int, int, int, NodeType[], int[])
    secondHashVal updates the Woodruff & Zemel second hashing value based on the tour insertion move.

tourHVwz(NodeType[], int[])
    tourHVwz method computes the Woodruff & Zemel hashing value from the sum of adjacent node id multiplication.

Constructors

HashMod

    public HashMod()

Methods

lookFor

    public static boolean lookFor(Hashtable daHashTab,
        int fhv,
        int shv,
        int cost,
        int tvl,
        int twPen,
        int loadPen,
        int lastIter)

        lookFor method searches for the current tour in the hashing structure, if the tour is found a true value for the boolean "found" is returned, if not found, the tour is added to the hashtable.

Parameters:
    daHashTab - hashtable object.
    fhv - First hashing value (objective function).
    shv - Second hashing value (Woodruff & Zemel).
    tourCost - Tour cost.
    twl - Travel time.
    twPen - Time window penalty.
    loadPen - Load overage penalty.
    lastIter - Iteration on which the tour was previously found.

Returns:
    returns true boolean value if the tour was previously found.

randWtWZ

    public static final int[] randWtWZ(int ZRANGE,
        int nc,
        int numnodes)

    randWtWZ method computes Woodruff & Zemel random weights between 1 & range for all nodes.
Parameters:
  ZRANGE - maximum weight value.
  nc - number of customers (targets).
  numnodes - total number of nodes.

Returns:
  returns integer array of "z" weights.

**tourHVwz**

```java
class TourHVwz {
    public static final int tourHVwz(NodeType tour[], int zArr[]) {
        // Method computes the Woodruff & Zemel hashing value from the sum of adjacent node id multiplication.
        return compute; // Implementation details
    }
}
```

Parameters:
  tour - tour node array to be processed.
  zArr - "z" array of random weights.

Returns:
  returns secondary hashing value function (thv).

**firstHashVal**

```java
class FirstHashVal {
    public static final int firstHashVal(int zT) {
        // Method assigns the primary hashing value. Currently, it assigns the objective function as the first hashing value (fhv). Method can be updated as desired.
        return compute; // Implementation details
    }
}
```

Parameters:
  zT - objective function value.

Returns:
  returns first hashing value (fhv).

**secondHashVal**

```java
class SecondHashVal {
    public static final int secondHashVal(int shv, int chI, int chD, NodeType tour[], int zArr[]) {
        // Method updates the Woodruff & Zemel second hashing value based on the tour insertion move.
        return compute; // Implementation details
    }
}
```

Parameters:
  shv - current tour hashing value.
  chI - node insertion position.
  chD - node insertion depth.
  tour - tour node array for processing.
  zArr - "z" array of random weights.

Returns:
  returns updated hashing value to reflect insertion.

**Class InFromKeybd**

```java
import java.lang.Object;

class InFromKeybd {
    // Class allows us to enter strings, integers, doubles and floats from the keyboard with a specified prompt.
    public class InFromKeybd {
        // Implementation details
    }
}
```

Version:
  v1.1 Feb 99
Author:
   Kevin P. O'Rourke, David M. Ryer

Constructor Index

InFromKeybd()

Method Index

keyDouble(String)
   keyDouble allows user to enter a double from the keyboard.

keyFloat(String)
   keyFloat allows user to enter a float from the keyboard.

keyInt(String)
   keyInt allows user to enter an integer from the keyboard.

keyString(String)
   keyString allows user to enter a string from the keyboard.

Constructors

InFromKeybd

   public InFromKeybd()

Methods

keyString

   public static final String keyString(String prompt)
   keyString allows user to enter a string from the keyboard.
   Parameters:
      prompt - Text prompt printed on screen.
   Returns:
      returns user entered string.

keyInt

   public static final int keyInt(String prompt)
   keyInt allows user to enter an integer from the keyboard.
   Parameters:
      prompt - Text prompt printed on screen.
   Returns:
      returns user entered integer.

keyDouble

   public static final double keyDouble(String prompt)
   keyDouble allows user to enter a double from the keyboard.
   Parameters:
      prompt - Text prompt printed on screen.
   Returns:
      returns user entered double.

keyFloat

   public static final float keyFloat(String prompt)
   keyFloat allows user to enter a float from the keyboard.
   Parameters:
      prompt - Text prompt printed on screen.
   Returns:
      returns user entered float.
Class KeyObj

java.lang.Object
   |-----KeyObj

public final class KeyObj
extends Object
KeyObj Class is used to access tour attributes in the hashtable for comparison.
Version:
v1.1 Mar 99
Author:
   Kevin P. O'Rourke, David M. Ryer
Constructor Index
KeyObj(int, int, int, int, int, int)
   Specified constructor.
Method Index
equals(KeyObj)
   Overloaded equals(), check only attribute fields.
hashCode()
   Overloaded hashCode method.
toString()
   toString changes a KeyObj to a string for use in the hashTable.
Constructors
KeyObj

public KeyObj(int fhv,
   int shv,
   int cost,
   int tvl,
   int twPen,
   int loadPen)
   Specified constructor. Values set as passed.
Methods
equals

public final boolean equals(KeyObj a)
   Overloaded equals(), check only attribute fields. Do not check first two data elements to keep
inline with hashCode overload.
Parameters:
a - element compared calling object.
Returns:
   returns true if objects are equal, false otherwise.
toString

public final String toString()
   toString changes a KeyObj to a string for use in the hashTable.
Returns:
   returns concatenated String.
Overrides:
   toString in class Object
hashCode
public final int hashCode()

Overloaded hashCode method. Note: if two objects are equal according to the equals method, then calling the hashCode method on each of the two objects must produce the same integer result.

Returns:

returns integer hashcode value.

Overrides:

hashCode in class Object

Class KeyToString

java.lang.Object

| +---- KeyToString

public class KeyToString

extends Object

KeyToString Class converts the attributes of tour to a concatenated string used as a key to the hashtable of tours.

Version:

v1.1 Mar 99

Author:

Kevin P. O'Rourke, David M. Ryer

Constructor Index

KeyToString()

Method Index

keyToString(int, int, int, int, int)

KeyToString Class converts the attributes of tour to a concatenated string used as a key to the hashtable of tours.

Constructors

KeyToString

public KeyToString()

Methods

keyToString

public static String keyToString(int fhv,

int shv,

int tourCost,

int tvl,

int twPen,

int loadPen)

KeyToString Class converts the attributes of tour to a concatenated string used as a key to the hashtable of tours.

Parameters:

fhv - First hashing value (objective function).
shv - Second hashing value (Woodruff & Zemel).
tourCost - Tour cost.
tv1 - Travel time.
twPen - Time window penalty.
loadPen - Load overage penalty.

Class MTSPTW

```
java.lang.Object
 +-----MTSPTW
```

public class MTSPTW
extends Object
MTSPTW is the main part that implements the multiple traveling salesperson problem with
time windows solve algorithm. This version calls the specific methods to read file input and generate the
appropriate time matrix.
Version:
v1.1 Mar 99

Author:
Kevin P. O'Rourke, David M. Ryer

Constructor Index

MTSPTW()

Method Index

main(String[])
    main executes MTSPTW problem.
swap(int, int)
    Swap allows generic swap of integers.

Constructors

MTSPTW

public MTSPTW()

Methods

swap

public static void swap(int a,
    int b)
    Swap allows generic swap of integers.
Parameters:
a - integer
b - integer
Returns:
returns void

main

public static void main(String argv[])
    main executes MTSPTW problem. Initializes global variables, calls methods to read data and wind
files, calls method to compute time matrix, calls tabu search method, writes output to file.
Parameters:
v - number of vehicles, overridden by file information
iters - number of iterations
integer - precision scaling factor
file - data file name, without extension (actual filename must end with .dat).
wind - file name, without extension (actual filename must end with .dat).
reroute - identifier. Use 111 (one one one) to specify reroute.

Class NoCycleOut

java.lang.Object
  |
  +-----NoCycleOut

public class NoCycleOut
extends Object
NoCycleOut is used as a package to output multiple fields from the method NoCycle.
Version: v1.1 Mar 99
Author: Kevin P. O'Rourke, David M. Ryer

Variable Index
ssltlc  cycle related variable.
tabuLen  cycle related variable.

Constructor Index
NoCycleOut()
  Default constructor.
NoCycleOut(int, int)
  Specified constructor.

Variables
ssltlc
  public int ssltlc
    cycle related variable.
tabuLen
  public int tabuLen
    cycle related variable.

Constructors
NoCycleOut
  public NoCycleOut()
    Default constructor. Assigns all values to zero.
NoCycleOut
  public NoCycleOut(int ssltlc,
                     int tabuLen)
    Specified constructor. Values set as passed.

Class NodeType
java.lang.Object
| +-----NodeType

public class NodeType
extends Object
NodeType defines the relevant information of each particular node.
Version:
v1.1 Feb 99
Author:
Kevin P. O'Rourke, David M. Ryer

Constructor Index

NodeType()
  Default constructor.
NodeType(int, int, int, int, int, int)
  Specified constructor.

Method Index

compPens(NodeType[], int)
  compPens computes the vehicle capacity overload and time window penalties.

copy()

countVeh(NodeType[])
  countVeh finds the number of vehicles being used in the current tour by counting
  the vehicle to demand transitions.

getEa()

getId()

getLa()

getLoad()

getM()

getQty()

getType()

getWait()

insert(NodeType[], int, int)
  insert allows the element designated by "chl" to be shifted by "chD" elements.

moveValTT(int, int, NodeType[], NodeType[], int[][]) [1]
  moveValTT computes the incremental change in the value of the travel time from
  the incumbent tour to the proposed neighbor tour, and computes the neighbor
  schedule parameters preparing for computation of penalty terms.

print()

printTour(NodeType[])

setId(int)

setLoad(int)

setQty(int)

setType(int)

setWait(int)

startTour(NodeType[], int[][], int, int)
  startTour will bubble sort the initial tour based on the average time window time.

sumWait(NodeType[])
  sumWait calculates the total "waiting" time in a particular tour by summing the wait
  values for each individual node.

swapInt(int, int)
  swapInt switches two integers

swapNode(NodeType[], int, int)
  swapNode allows the node array elements "a" and "b" to be swapped in the Node Array
  "z".

tourSched(int, NodeType[], int[][])

70
Method `tourSched` should be called with the syntax `tourLen = tourSched(nodeArray, time)` from the `orderStartingTour` method.

**Constructors**

**NodeType**

```java
class NodeType {
    public NodeType() {
        // Default constructor. Assigns all values to zero.
    }

    public NodeType(int id,
                    int ea,
                    int la,
                    int qty,
                    int type,
                    int wait,
                    int load)
        Specified constructor. Values set as passed.
}
```

**Methods**

**copy**

```java
public final NodeType copy() {
    // Method to copy the current instance of NodeType.
}
```

**swapInt**

```java
public static final void swapInt(int a,
                                 int b)
    Method `swapInt` switches two integers
```

**swapNode**

```java
public static final NodeType[] swapNode(NodeType z[],
                                        int a,
                                        int b)
    Method `swapNode` allows the node array elements "a" and "b" to be swapped in the Node Array "z".
```

Parameters:
- `z` - node array to be updated.
- `a` - element to be swapped.
- `b` - element to be swapped.

Returns:
- returns updated node array.

**insert**

```java
public static final NodeType[] insert(NodeType z[],
                                      int chi,
                                      int chD)
    Method `insert` allows the element designated by "chi" to be shifted by "chD" elements. chD may be positive or negative.
```

Parameters:
- `z` - node array to be updated.
- `chi` - location of node to be moved.
- `chD` - depth of move.

Returns:
- returns updated node array.
countVeh

public static final int countVeh(NodeType tour[])

Method countVeh finds the number of vehicles being used in the current tour by counting the vehicle to demand transitions.
Parameters:
    tour - node array to be processed.
Returns:
    returns integer number of vehicles used in the tour.

sumWait

public static final int sumWait(NodeType tour[])

Method sumWait calculates the total "waiting" time in a particular tour by summing the wait values for each individual node.
Parameters:
    tour - node array to be processed.
Returns:
    returns integer value of total wait time in the tour.

compPens

public static final VrpPenType compPens(NodeType tour[], int capacity)

compPens computes the vehicle capacity overload and time window penalties.
Parameters:
    tour[] - current tour used to calculate penalties.
    capacity - maximum vehicle load.
Returns:
    returns the VrpPenType object which the method was called on with updated values.

tourSched

public static final int tourSched(int is, NodeType tour[],
                                 int time[][])

Method tourSched should be called with the syntax tourLen = tourSched(nodeArray, time) from the orderStartingTour method. This will use the listing of nodes to return the new tourLen value (tour duration). Additionally, the nodeArray will be updated to reflect the new arrival and departure times.
Parameters:
    is - insertion/starting location for computation of schedule.
    tour - node array to be processed.
    time - time matrix used to determine schedule.
Returns:
    returns integer total tour duration. Updates tour node array as appropriate.

startTour

public static final int startTour(NodeType tour[],
                                 int time[][],
                                 int nc,
                                 int nv)

Method startTour will bubble sort the initial tour based on the average time window time. No swap is made if the move would violate strong time window infeasibility.
Parameters:
    tour - node array to be processed.
    time - time matrix used to determine schedule.
    nc - number of customers.
nv - number of vehicles.

Returns:
returns integer total tour duration. Updates tour node array as appropriate.

**getId**

```java
public final int getId()
```

**getEa**

```java
public final int getEa()
```

**getLa**

```java
public final int getLa()
```

**getQty**

```java
public final int getQty()
```

**getType**

```java
public final int getType()
```

**getWait**

```java
public final int getWait()
```

**getLoad**

```java
public final int getLoad()
```

**getM**

```java
public final double getM()
```

**setId**

```java
public final void setId(int id)
```

**setWait**

```java
public final void setWait(int wait)
```

**setType**

```java
public final void setType(int type)
```

**setQty**

```java
public final void setQty(int qty)
```

**setLoad**

```java
public final void setLoad(int load)
```

**print**

```java
public final void print()
```

**printTour**

```java
public static final void printTour(NodeType tour[])
```

**moveValTT**

```java
public static int moveValTT(int i,
    int d,
    NodeType tour[],
```

```java
)
```

73
NodeType mbtortour[],
   int time[][],

Method moveValTT computes the incremental change in the value of the travel time from the
incumbent tour to the proposed neighbor tour, and computes the neighbor schedule parameters
preparing for computation of penalty terms.

Parameters:
   i - node position.
   d - move depth.
   tour - incumbent tour node array to be processed.
   nbtour - neighbor tour node array to be processed.
   time - time matrix used to determine schedule.

Returns:
   returns integer move value which is the resultant change in the objective function resulting from
   the proposed move.

See Also:
   compPens

Class PrintCalls

java.lang.Object
   |
   +-----PrintCalls

public class PrintCalls
extends Object PrintCalls is to display on the screen initial values and rts steps as required.

Version:
   v1.1 Mar 99

Author:
   Kevin P. O'Rourke, David M. Ryer

Constructor Index

PrintCalls()

Method Index

printInitVals(int, int, int, double, String)
rtsStepPrint(int, int, int, int, int, int, int)

Constructors

PrintCalls

   public PrintCalls()

Methods

printInitVals

   public static void printInitVals(int nv,
                                       int iters,
                                       int numcycles,
                                       double factor,
                                       String file)

rtsStepPrint

   public static void rtsStepPrint(int id,
                                       int i,
int d,
int k,
int moveVal,
int totNbrPen,
int tabu,
int numnodes)

Class PrintFlag

java.lang.Object

|-----PrintFlag

public class PrintFlag
extends Object
PrintFlag contains all print out flags as boolean attributes.
Version:
  v1.1 Mar 99
Author:
  Kevin P. O'Rourke, David M. Ryer

Variable Index

  cyclePrint
    print flag.
  iterPrint
    print flag.
  loadPrint
    print flag.
  movePrint
    print flag.
  startPrint
    print flag.
  stepLoopPrint
    print flag.
  stepPrint
    print flag.
  timePrint
    print flag.
  twrdPrint
    print flag.

Constructor Index

PrintFlag()
  Default PrintFlag constructor sets all to "true".
PrintFlag(boolean)
  Additional PrintFlag constructor allows specification of either "true" or "false".

Variables

movePrint

  public boolean movePrint
    print flag.

startPrint

  public boolean startPrint
    print flag.
timePrint

public boolean timePrint
    print flag.

stepPrint

public boolean stepPrint
    print flag.

stepLoopPrint

public boolean stepLoopPrint
    print flag.

twrdPrint

public boolean twrdPrint
    print flag.

cyclePrint

public boolean cyclePrint
    print flag.

iterPrint

public boolean iterPrint
    print flag.

loadPrint

public boolean loadPrint
    print flag.

Constructors

PrintFlag

public PrintFlag()
    Default PrintFlag constructor sets all to "true".

PrintFlag

public PrintFlag(boolean set)
    Additional PrintFlag constructor allows specification of either "true" or "false".

Class ReactTabuObj

java.lang.Object
    |   
    +----ReactTabuObj

public class ReactTabuObj
extends Object ReactTabuObj class contains the method to perform the reactive tabu search.
Version:
    v1.1 Mar 99
Author:
    Kevin P. O'Rourke, David M. Ryer
Constructor Index

ReacTabuObj()

Method Index

search(double, double, double, double, int, int, int, int, int, int, int, int, int, int, int, int, VrpPenType, int[][], PrintFlag, int, int, int, int, int, int, int, int, int, int, int, int, int, int, NodeType[], NodeType[], NodeType[])

ReacTabuObj steps through iterations of the reactive tabu search.

Constructors

ReacTabuObj

public ReacTabuObj()

Methods

search

public static SearchOut search(double TWPEN, double LDPEN, double INCREASE, double DECREASE, int HTSIZE, int CMAX, int ZRANGE, int DEPTH, int capacity, int minTL, int maxTL, int tabuLen, int iters, int nc, int numnodes, VrpPenType tourPen, int time[][]), PrintFlag printFlag, int tourCost, int penTrav, int totPenalty, int tvl, int bfTourCost, int btt, int bfn, int bfiter, int bestCost, int bestTT, int bestnv, int bestTime, int bestTimeP, int bestiter, int numfeas, NodeType tour[], NodeType bestTour[], NodeType bestTourP[])

ReacTabuObj steps through iterations of the reactive tabu search. This method will perform tabu search for VRP with capacity as well as TSP without capacity.

Returns:
returns packaged output object.

Class ReadFile

java.lang.Object
    ├──-----ReadFile

class ReadFile
extends Object
ReadFile Class reads appropriate data from a text file. Methods exist to read specific data
types (file format must be known in advance).
Version:
    v1.1 Mar 99
Author:
    Kevin P. O'Rourke, David M. Ryer

Constructor Index
isplay
Method Index
assignInputFile(String)
    assignInputFile sets up the FileInputStream.
readNextDouble(StreamTokenizer)
    readNextString method gets the next token and returns it as a double.
readNextInt(StreamTokenizer)
    readNextString method gets the next token and returns it as an integer.
readNextString(StreamTokenizer)
    readNextString method gets the next token and returns it as a string.

Constructors

ReadFile

class ReadFile()

Methods

assignInputFile

public static final FileInputStream assignInputFile(String filename)
    assignInputFile sets up the FileInputStream.

readNextString

public static final String readNextString(StreamTokenizer st)
    readNextString method gets the next token and returns it as a string.
Parameters:
    st - string tokenizer.
Returns:
    returns next string from file.

readNextDouble

public static final double readNextDouble(StreamTokenizer st)
    readNextString method gets the next token and returns it as a double.
Parameters:
    st - string tokenizer.
Returns:
returns next double from file.

**readNextInt**

```java
public static final int readNextInt(StreamTokenizer st)
```
readNextString method gets the next token and returns it as a integer.

Parameters:
- st - string tokenizer.

Returns:
- returns next integer from file.

**Class SearchOut**

```java
java.lang.Object
   |-----SearchOut
```

public class SearchOut extends Object SearchOut is used as a package to output multiple information from the Search method in ReactTabuObj.

Version:
- v1.1 Mar 99

Author:
- Kevin P. O'Rourke, David M. Ryer

See Also:
- Search

**Variable Index**

- bestCost
- bestiter
- bestnv
- bestTime
- bestTour
- bestTT
- bfCost
- bfiter
- bfnv
- bfTime
- bfTour
- bfTT
- numfeas
- penTray
- totPenalty
- tour
- tourCost
- tourPen
- tvl

**Constructor Index**

```java
SearchOut()
```
Default constructor.

```java
SearchOut(int, int, int, int, int, int, int, int, int, int, int, int, int, int, VrpPenType, NodeType[],
NodeTyope[], NodeType[])
```
Specified constructor.
Variables

totPenalty

public int totPenalty

penTrav

public int penTrav

tourCost

public int tourCost

bfiter

public int bfiter

bfCost

public int bfCost

bfTT

public int bfTT

bestnv

public int bestnv

bestiter

public int bestiter

bestCost

public int bestCost

bestTT

public int bestTT

bfnv

public int bfnv

bfTime

public int bfTime

bestTime

public int bestTime

tvl

public int tvl

numfeas

public int numfeas

tourPen

public VrpPenType tourPen

tour

public NodeType tour[]
bfTour

    public NodeType bfTour[]

bestTour

    public NodeType bestTour[]

Constructors

SearchOut

    public SearchOut()
    Default constructor. Assigns all values to zero.

SearchOut

    public SearchOut(int totPenalty,
                     int penTrav,
                     int tourCost,
                     int bfilter,
                     int bfCost,
                     int bTT,
                     int bestnv,
                     int bestiter,
                     int bestCost,
                     int bestTT,
                     int bfnv,
                     int bftime,
                     int bestTime,
                     int tvl,
                     int numfeas,
                     VrpPenType tourPen,
                     NodeType tour[],
                     NodeType bfTour[],
                     NodeType bestTour[])
    Specified constructor. Values set as passed.

Class StartPenBestOut

java.lang.Object

    +--------StartPenBestOut

public class StartPenBestOut
extends Object
StartPenBestOut is used as a package to output multiple penalty information from method startPenBest.
Version:
    v1.1 Mar 99
Author:
    Kevin P. O'Rourke, David M. Ryer

Variable Index

    bestCost
    Penalty related value.

    bestiter
    Penalty related value.
bestny
Penalty related value.
bestTime
Penalty related value.
bestTour
Saved tour.
bestTT
Penalty related value.
bCost
Penalty related value.
biter
Penalty related value.
bnev
Penalty related value.
bftime
Penalty related value.
bftour
Saved tour.
bftt
Penalty related value.
penTrav
Penalty related value.
totPenalty
Penalty related value.
tourCost
Penalty related value.
tourPen
Tour penalty values.

Constructor Index

StartPenBestOut()
Default constructor.
StartPenBestOut(int, int, int, int, int, int, int, int, int, int, int, VrpPenType, NodeType[],
NodeType[]) Specified constructor.

Variables
totPenalty

public int totPenalty
Penalty related value.
penTrav

public int penTrav
Penalty related value.
tourCost

public int tourCost
Penalty related value.
biter

public int biter
Penalty related value.
bCost


public int bfCost
    Penalty related value.

bfTT
    public int bfTT
    Penalty related value.

bestnv
    public int bestnv
    Penalty related value.

bestiter
    public int bestiter
    Penalty related value.

bestCost
    public int bestCost
    Penalty related value.

bestTT
    public int bestTT
    Penalty related value.

bfnv
    public int bfnv
    Penalty related value.

bfTime
    public int bfTime
    Penalty related value.

bestTime
    public int bestTime
    Penalty related value.

tourPen
    public VrpPenType tourPen
    Tour penalty values.

bfTour
    public NodeType[] bfTour[]
    Saved tour.

bestTour
    public NodeType bestTour[]
    Saved tour.

Constructors
StartPenBestOut
    public StartPenBestOut()
    Default constructor. Assigns all values to zero.

StartPenBestOut
public StartPenBestOut(int totPenalty,
    int penTrav,
    int tourCost,
    int bfilter,
    int bfCost,
    int bfTT,
    int bestNV,
    int bestIter,
    int bestCost,
    int bestTT,
    int bfnv,
    int bftime,
    int bestTime,
    VrppenType tourPen,
    NodeType bftour[],
    NodeType besttour[])

Specified constructor. Values set as passed.

Class StartTourObj

java.lang.Object
   |-------StartTourObj

public class StartTourObj
extends Object StartTourObj class begins timing, computes an initial schedule and initial tour cost (Tour Cost = Travel time + Waiting Time + Penalty Term), computes the initial hashing values: Z(i) and thv(i), and produces a tour based on a sort of increasing avg time windows at each node. The customers are ordered by increasing avg time window value, and the nv vehicle nodes are appended to the end of the tour.

Constructor Index
StartTourObj()

Method Index
startPenBest(int, int, int, NodeType[], double, double, int, int, int, VrppenType, int, int, int, int, int, int, Node[][], NodeType[][], Node[][])
startPenBest initializes "best" values and their times.

Constructors
StartTourObj

   public StartTourObj()

Methods
startPenBest

   public static StartPenBestOut startPenBest(int numnodes,
   int tvl,
   int tourLen,
   NodeType tour[],
   double TWPen,
   double LDpEN,
   int capacity,
   int totPenalty,
   int penTrav,
int tourCost,
VrpPenType tourPen,
int bfilter,
int bftourCost,
int bftt,
int bfnv,
int bestiter,
int bestCost,
int bestTt,
int bestnv,
int besttimeF,
int bestTime,
NodeType bestTour[],
NodeType bestTourF[])

startPenBest initializes "best" values and their times. Computes cost of initial tour as tour length
with added penalty for infeasibilities.

Returns:
returns StartPenBestOut wrapper object for multiple values.

Class TabuMod

java.lang.Object
   |
   +----TabuMod

public class TabuMod
extends Object TabuMod Class contains methods used in the TabuSearch. countVeh calculates the number
of vehicles used in the current tour. noCycle updates the search parameters if tour is not found in the
hashtable. cycle updates the search parameters if tour is found in the hashtable. moveValTT computes the
incremental change in the value of the travel time.

Version:
v1.1 Mar 99

Author:
   Kevin P. O'Rourke, David M. Ryer

Constructor Index

TabuMod()

Method Index

countVehicles(NodeType[])
countVeh method calculates the number of vehicles used in the current tour by counting the
number of vehicle (type 2) to demand (type 1) transitions.
cycle(ValueObj, double, int, int, double, int, int, PrintFlag)
cycle method updates the search parameters if the incumbent tour is found in the hashing structure.
moveValTT(int, int, NodeType[], NodeType[], int[][])
Method moveValTT computes the incremental change in the value of the travel time from the
incumbent tour to the proposed neighbor tour, and computes the neighbor schedule parameters
preparing for computation of penalty terms.

noCycle(double, int, double, int, PrintFlag)
noCycle method updates the search parameters if the incumbent tour is not found in the hashing
structure.

Constructors

TabuMod
public TabuMod()

Methods

countVehicles

public static final int countVehicles(NodeType tour[])
    countVeh method calculates the number of vehicles used in the current tour by counting the
    number of vehicle (type 2) to demand (type 1) transitions.
Parameters:
    tour - node array to be processed.
Returns:
    returns integer number of vehicles used in the tour.

noCycle

public static NoCycleOut noCycle(double DECREASE,
                                    int minTL,
                                    double mavg,
                                    int ssltlc,
                                    int tabuLen,
                                    PrintFlag printFlag)
    noCycle method updates the search parameters if the incumbent tour is not found in the hashing
    structure.
Parameters:
    DECREASE - adjustable scaling factor to reduce tabu length.
    minTL - minimum tabu length.
    mavg - moving average between cycles.
    ssltlc - steps since last tabu length change.
    tabuLen - current tabu length.
    printFlag - option to print cycle information.
Returns:
    returns noCycleOut wrapped object.
cycle

public static CycleOut cycle(ValueObj matchPtr,
                                   double INCREASE,
                                   int maxTL,
                                   int CYMAX,
                                   int k,
                                   double mavg,
                                   int ssltlc,
                                   int tabuLen,
                                   PrintFlag printFlag)
    cycle method updates the search parameters if the incumbent tour is found in the hashing structure.
Parameters:
    matchPtr - matched information for previously found identical tour
    INCREASE - adjustable scaling factor to increase tabu length
    maxTL - maximum tabu length
    CYMAX - maximum allowable cycle frequency
    k - current iteration
    mavg - moving average between cycles.
    ssltlc - steps since last tabu length change.
    tabuLen - current tabu length.
    printFlag - option to print cycle information.
Returns:
    returns cycleOut wrapped object.
moveValTT

public static int moveValTT(int i,
    int d,
    NodeType tour[],
    NodeType nbrtour[],
    int time[][])

Method moveValTT computes the incremental change in the value of the travel time from the
incumbent tour to the proposed neighbor tour, and computes the neighbor schedule parameters
preparing for computation of penalty terms.

Parameters:
    i - node position.
    d - move depth.
    tour - incumbent tour node array to be processed.
    nbrtour - neighbor tour node array to be processed.
    time - time matrix used to determine schedule.

Returns:
    returns integer move value which is the resultant change in the objective function resulting from
    the proposed move.

See Also:
    compPens

Class TimeMatrixObj

java.lang.Object
   |
   +----+TimeMatrixObj

public class TimeMatrixObj
extends Object TimeMatrixObj contains methods to calculate the distance/time matrix based on the
problem parameters.

Version:
    v1.1 Mar 99

Author:
    Kevin P. O'Rourke, David M. Ryer

Constructor Index

TimeMatrixObj()

Method Index

readNC(String)
    readNC is used to read from the first token from the input file (the number of customers (nc)).

readNV(String)
    readNV is used to read from the second token from the input file (the number of vehicles (nv)).

readTSPTW(double, int, int, String, CoordType[], int[])
    readTSPTW reads in the geographical coordinates and time window file and calculates the time
    between each node.

readTSPTWddepot(double, int, int, String, CoordType[], int[])
    readTSPTWddepot reads in the geographical coordinates, load quantity, service time, and time
    window information associated with depot and customer locations from the input file.

timeMatrix(int, int, double, int, CoordType[], int[])
    timeMatrix computes simple two-dimensional time/distance matrix.

timeMatrixDepot(int, int, double, int, CoordType[], int[])
    timeMatrixDepot computes the two-dimensional array used as the "time" matrix.
Constructors

TimeMatrixObj

public TimeMatrixObj()

Methods

readNC

public static int readNC(String filein)
readNC is used to read from the first token from the input file (the number of customers (nc)).
Parameters:
filein -- name of input file
Returns:
returns nc number of customers

readNV

public static int readNV(String filein)
readNV is used to read from the second token from the input file (the number of vehicles (nv)).
Parameters:
filein -- name of input file
Returns:
returns nv number of vehicles

readTSPTW

public static NodeType[] readTSPTW(double factor,
int nv,
int nc,
String filein,
CoordType coord[],
int s[])

readTSPTW reads in the geographical coordinates and time window file and calculates the time
between each node
Parameters:
factor -- integer scaling factor used to increase precision.
v -- number of aircraft available (vehicles).
c -- number of targets/route points (customers).
filein -- name of input file.
coord -- blank array where coordinates will be stored upon method completion.
s -- blank array where service times will be stored upon method completion.
Returns:
returns the tour array reflecting file data.

readTSPTWdepot

public static NodeType[] readTSPTWdepot(double factor,
int nv,
int nc,
String filein,
CoordType coord[],
int s[])

readTSPTWdepot reads in the geographical coordinates, load quantity, service time, and time
window information associated with depot and customer locations from the input file. This
information is returned as a tour array.
Parameters:
factor -- integer scaling factor used to increase precision.
v -- number of aircraft available (vehicles).
nc -- number of targets/route points (customers).
filein -- name of input file.
coord -- blank array where coordinates will be stored upon method completion.
s -- blank array where service times will be stored upon method completion.

Returns:
returns the tour array reflecting file data.

timeMatrix

public static int[][] timeMatrix(int nc,
        int gamma,
        double factor,
        int numnodes,
        CoordType coord[],
        int s[])

timeMatrix computes simple two-dimensional time/distance matrix.

Parameters:
nc -- number of targets/route points (customers).
gamma -- additional vehicle usage penalty (set to ZERO only).
factor -- integer scaling factor used to increase precision.
coord -- blank array where coordinates will be stored upon method completion.
s -- blank array where service times will be stored upon method completion.

Returns:
returns the time matrix specific to the problem.

timeMatrixDepot

public static int[][] timeMatrixDepot(int nc,
        int gamma,
        double factor,
        int numnodes,
        CoordType coord[],
        int s[])

timeMatrixDepot computes the two-dimensional array used as the "time" matrix. This time matrix
contains the travel times between respective nodes, general setup for multiple depot problem.

Parameters:
nc -- number of targets/route points (customers).
gamma -- additional vehicle usage penalty (set to ZERO only).
factor -- integer scaling factor used to increase precision.
coord -- blank array where coordinates will be stored upon method completion.
s -- blank array where service times will be stored upon method completion.

Returns:
returns the time matrix specific to the problem.

Class Timer

defined as

class Timer extends Object

timer1, timer2, etc.

| +---- Timer

public class Timer
extends Object Timer Class is used to time overall computation time.
Version:
v1.1 Mar 99
Author:
   Kevin P. O'Rourke, David M. Ryer

89
Variable Index

**endTime**
end time.

**startTime**
begin time.

**totalSeconds**
duration of run.

Constructor Index

**Timer()**
Default constructor.

Method Index

**endTime()**
endTime assigns end time.

**startTime()**
startTime assigns start time.

**totalSeconds()**
totalSeconds returns duration.

Variables

**startTime**

```java
public long startTime
begin time.
```

**endTime**

```java
public long endTime
end time.
```

**totalSeconds**

```java
public long totalSeconds
duration of run.
```

Constructors

**Timer**

```java
public Timer()
Default constructor. Assigns all values to zero.
```

Methods

**startTime**

```java
public long startTime()
startTime assigns start time.
Returns:
returns start time.
```

**endTime**

```java
public long endTime()
endTime assigns end time.
Returns:
returns end time.
```

**totalSeconds**
public long totalSeconds()
    totalSeconds returns duration.
Returns:
    returns duration.

Class TsptwPen

java.lang.Object
    \|-- MTSPTW
       \|-- TsptwPen

public class TsptwPen
extends MTSPTW tsptwPen class: Given the TW and load penalties, this procedure personalizes the penalties to the mTSPTW; Computes tourCost of tour as tour length + scaled penalty for infeasibilities.

Constructor Index

TsptwPen()

Method Index

TsptwPen(int, NodeType[], VrpPenType, double, double, int, int, int, int)
    tsptwPen method uses the TW and load penalties to computes tourCost of tour as tour length + scaled penalty for infeasibilities.

TsptwPenNormalized(int, NodeType[], VrpPenType, double, double, int, int, int, int)
    tsptwPenNormalized method uses the TW and load penalties to computes tourCost of tour as tour length + scaled penalty for infeasibilities.

Constructors

TsptwPen

public TsptwPen()

Methods

TsptwPen

public static final TsptwPenOut tsptwPen(int tourLen,
    NodeType tour[],
    VrpPenType tourPen,
    double TWPEN,
    double LDPEN,
    int totPenalty,
    int tourCost,
    int penTrav,
    int tvl)
    tsptwPen method uses the TW and load penalties to computes tourCost of tour as tour length + scaled penalty for infeasibilities. This method is used with the absolute penalty factors.

Parameters:
    tourLen - tour duration.
    tour - node array to be processed.
    tourPen - current tour penalty value.
    TWPEN - time window penalty factor.
    LDPEN - load overage penalty factor.
    totPenalty - sum total penalties.
tourCost - total tour cost.
penTrav - travel time penalty.
tvl - travel duration.

Returns:
returns wrapped multiple objects.

tsttwPenNormalized

public static final TsptwPenOut tsptwPenNormalized(int tourLen,
            NodeType tour[],
            VrpPenType tourPen,
            double TWPen,
            double LDpen,
            int totPenalty,
            int tourCost,
            int penTrav,
            int tvl)

TstptwPenNormalized method uses the TW and load penalties to computes tourCost of tour as tour length + scaled penalty for infeasibilities. This method is uses penalty factors of one and is called when the insertion move is made. Penalty values are then comparable from iteration to iteration.

Parameters:
tourLen - tour duration.
tour - node array to be processed.
tourPen - current tour penalty value.
TWPen - time window penalty factor (IGNORED, set to 1).
LDpen - load overage penalty factor (IGNORED, set to 1).
totPenalty - sum total penalties.
tourCost - total tour cost.
penTrav - travel time penalty.
tvl - travel duration.

Returns:
returns wrapped multiple objects.

Class TsptwPenOut

java.lang.Object
    | +-----TsptwPenOut

public class TsptwPenOut
extends Object TsptwPenOut is used as a package to output multiple penalty information from class TsptwPen.
Version:
    v1.1 Mar 99

Author:
Kevin P. O'Rourke, David M. Ryer

Variable Index

penTrav  Penalty related value.
totPenalty  Penalty related value.
tourCost  Penalty related value.
tvl  Penalty related value.
Constructor Index

TsptwPenOut()
Default constructor.

TsptwPenOut(int, int, int, Int)
Specified constructor.

Variables
totPenalty

public int totPenalty
Penalty related value.
tourCost

public int tourCost
Penalty related value.
penTrav

public int penTrav
Penalty related value.
tvl

public int tvl
Penalty related value.

Constructors

TsptwPenOut

public TsptwPenOut()
Default constructor. Assigns all values to zero.

TsptwPenOut

public TsptwPenOut(int totPenalty,
int tourCost,
int penTrav,
int tvl)
Specified constructor. Values set as passed.

Class TwBestTTOut

java.lang.Object
 |
+-----TwBestTTOut

public class TwBestTTOut
extends Object TwBestTTOut is used as a package to output multiple information from the TWBestTTOut method.
Version:
v1.1 Mar 99
Author:
Kevin P. O'Rourke, David M. Ryer

Variable Index

bestCost
best tour related value.

bestIter
best tour related value.

bestNv
best tour related value.

bestTime
best tour related value.

bestTour
best tour related value.

bestTT
best tour related value.

bfCost
best tour related value.

bfIter
best tour related value.

bfNv
best tour related value.

bfTime
best tour related value.

bfTour
best tour related value.

bfTT
best tour related value.

Constructor Index

TwBestTTOut()
Default constructor.

TwBestTTOut(int, int, int, int, int, int, int, int, int, Node[][], Node[][]) Specified constructor.

Variables

bfCost

public int bfCost
best tour related value.

bfTT

public int bfTT
best tour related value.

bfNv

public int bfNv
best tour related value.

bfIter

public int bfIter
best tour related value.

bestCost

public int bestCost
best tour related value.

bestTT

public int bestTT
best tour related value.
bestnv

public int bestnv
    best tour related value.

bestiter

public int bestiter
    best tour related value.

bfTime

public int bfTime
    best tour related value.

bestTime

public int bestTime
    best tour related value.

bfTour

public NodeType bfTour[]
    best tour related value.

bestTour

public NodeType bestTour[]
    best tour related value.

Constructors

TwBestTTOut

public TwBestTTOut()
    Default constructor. Assigns all values to zero.

TwBestTTOut

public TwBestTTOut(int bfCost,
                   int bfTT,
                   int bfnv,
                   int bfiter,
                   int bestCost,
                   int bestTT,
                   int bestnv,
                   int bestiter,
                   int bfTime,
                   int bestTime,
                   NodeType bfTour[],
                   NodeType bestTour[])

Specified constructor. Values set as passed.

Class ValueObj

java.lang.Object
    |
    +----ValueObj

public final class ValueObj
extends Object ValueObj Class is used to store tour attributes in the hashtable for comparison.
Version:  v1.1 Mar 99
Author: Kevin P. O'Rourke, David M. Ryer

Constructor Index
ValueObj(int, int, int, int, int, int)
    Specified constructor.

Method Index
equals(ValueObj)
    Overloaded equals(), check only attribute fields.
hashCode()
    Overloaded hashCode method.
toString()
    toString changes a ValueObj to a string for use in the hashTable.

Constructors
ValueObj

    public ValueObj(int fhv,
                    int shv,
                    int tourCost,
                    int tvl,
                    int twPen,
                    int loadPen,
                    int lastIter)
        Specified constructor. Values set as passed.

Methods
equals

    public final boolean equals(ValueObj a)
        Overloaded equals(), check only attribute fields. Do not check first two data elements to keep
        inline with hashCode overload.
Parameters:
    a - element compared calling object.
Returns:
    returns true if objects are equal, false otherwise.

toString

    public final String toString()
    toString changes a ValueObj to a string for use in the hashTable.
Returns:
    returns concatenated String.
Overrides:
    toString in class Object

hashCode

    public final int hashCode()
    Overloaded hashCode method. Note: if two objects are equal according to the equals method, then
    calling the hashCode method on each of the two objects must produce the same integer result. Do
    not checking first two data elements because of size limitations of Integer.
Returns:
    returns integer hashcode value.
Overrides:
   hashCode in class Object

Class VrpPenType

java.lang.Object
   |
   +++ VrpPenType

public class VrpPenType
extends Object
VrpPenType class provides the object structure for load and time window penalties.
Version: v1.1 Feb 99
Author: Kevin P. O'Rourke, David M. Ryer

Constructor Index
VrpPenType()
   Default constructor.
VrpPenType(int, int)
   Specified constructor.
VrpPenType(int, int, int)
   Specified constructor.

Method Index
compPens(NodeType[], int)
   compPens computes the vehicle capacity overload and time window penalties.

Constructors
VrpPenType

   public VrpPenType()
      Default constructor. Assigns all values to zero.
VrpPenType

   public VrpPenType(int tw,
      int ld)
      Specified constructor. Values set as passed.
VrpPenType

   public VrpPenType(int tw,
      int ld,
      int nvu)
      Specified constructor. Values set as passed.

Methods
compPens

   public final VrpPenType compPens(NodeType tour[],
      int capacity)
      compPens computes the vehicle capacity overload and time window penalties.
Parameters:
tour[] - current tour used to calculate penalties.
capacity - maximum vehicle load.
Returns:
returns the VrpPenType object which the method was called on with updated values.

Class WindAdjust

java.lang.Object
   
   public class WindAdjust
extends Object WindAdjust will provides the adjusted ground speed given the desired heading from
location A to location B, and the wind heading.
Version:
v1.1 Feb 99
Author:
   Kevin P. O'Rourke, David M. Ryer

Constructor Index

Constructor

WindAdjust()

Method Index

groundSpeed(double, double, double, double)
   groundSpeed method returns the ground speed given the heading between points, the wind
heading, the wind speed, and the aircraft's airspeed.

groundSpeedAF(double, double, double, double)
   groundSpeedAF is an experimental method that uses a different formula.

Constructors

WindAdjust

public WindAdjust()

Methods

groundSpeed

public static final double groundSpeed(double headingAtoB,
   double windDir,
   double airSpeed,
   double windSpeed)
   groundSpeed method returns the ground speed given the heading between points, the wind
heading, the wind speed, and the aircraft's airspeed.
Parameters:
headingAtoB - heading between points in degrees.
windDir - wind heading in degrees.
airSpeed - aircraft air speed in knots.
windSpeed - wind speed in knots.
Returns:
returns ground speed in knots.
Bibliography


Vita

Major David M. Ryer was born in Wilkes-Barre, Pennsylvania, on 22 December 1965. He graduated from Bensalem High School in 1983 and entered the United States Air Force Academy. He graduated with a Bachelor of Science degree in Electrical Engineering and received his commission in May 1987. Upon graduation from Undergraduate Pilot Training at Williams AFB in July 1988, he was assigned as a KC-135 pilot at Castle AFB, CA. In August 1997 he entered the Graduate School of Engineering, Air Force Institute of Technology.

Permanent Address: 1511 Barnswallow Drive
Bensalem, Pa 19020
Implementation of the Metaheuristic Tabu Search in Route Selection for Mobility Analysis Support System

Air Force Institute of Technology
2950 P Street
WPAFB, OH 45433

Major Robert Brigantic
AMC/XPY
402 Scott Drive
Unit 3L3
SCOTT AFB IL 62225

Approved for public release; distribution unlimited

This thesis employs a reactive tabu search heuristic implemented in the Java programming language to solve a real world variation of the vehicle routing problem with the objective of providing quality routes to Mobility Analysis Support System (MASS). MASS is a stochastic simulation model used extensively by Air Mobility Command (AMC) to analyze strategic airlift capabilities and future procurement decisions. This dynamic real world problem of strategic and tactical airlift possesses a number of side constraints such as vehicle capacities, route length and time windows in a sizeable network with multiple depots and a large fleet of heterogeneous vehicles. Finding optimal solutions to this problem is currently not practical. Currently, MASS requires all possible routes used in its simulation to be manually selected. As a result, the route selection process is a tedious and time consuming process that relies on experience and past performance of the model to obtain quality routes for the mobility system.

Tabu Search, Vehicle Routing Problem, Java, MASS, Mobility Modeling, Mobility Analysis Support System

UNCLASSIFIED

UNCLASSIFIED

UNCLASSIFIED