Vehicle Minimization for the Multimodal Pickup and Delivery Problem with Time Windows

Benjamin A . Clapp

VEHICLE MINIMIZATION FOR THE MULTIMODAL PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

THESIS

Benjamin Clapp

AFIT-ENS-13-M-03

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-13-M-03

VEHICLE MINIMIZATION FOR THE MULTIMODAL PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Benjamin A. Clapp, BS

March 2013

AFIT-ENS-13-M-03

VEHICLE MINIMIZATION FOR THE MULTIMODAL PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

Benjamin Clapp

Approved:

Jeffery Weir, PhD (Advisor) Date

Raymond Hill, PhD (Reader) Date

Accepted:

H. Ries, PhD Date

Dean, Graduate School of Engineering and Management

AFIT-ENS-13-M-03

Abstract

This thesis develops an algorithm to address a special case of the Vehicle Routing Problem. The algorithm developed is decompositional with two components. The first component is based upon Dijkstra's algorithm and is used to simplify the routing component of processing. The second component is based upon the priority rule heuristics used in scheduling job shop problems for parallel machines.

The VRP solved is subject to time windows and capacity constraints on vehicles and offloading. The VRP is multimodal. The objective function for the problem is the sum of all vehicles used, multiplied by their respective cost modifiers. Shipments are required to travel entirely on a single mode.

The data input consists of a network and shipping requirements. The network is subjected to Dijkstra's. Dijkstra's returns a simplified network of shortest paths. This simplified network, along with the shipping requirements, is subjected to the scheduling heuristic. The heuristic assigns as many of the shipments as possible away from the currently minimizing mode. This determines which shipments must be processed on the minimizing mode. It determines how many vehicles are required to carry those shipments. Finally, any remaining capacity is assigned. This process is repeated for each mode.

Acknowledgements

I would like to thank Dr. Weir for guidance throughout the development of this thesis and especially for his aid in paring away what was unimportant and retaining what was essential in the problem. I would also like to thank my sponsor, David Longhorn, for providing the problem and for his insight into it.

I would also like to thank my family and my wife, whose support helped me through this process. Finally, I would like to thank the Air Force Institute of Technology and Transportation Command, for making this research possible.

Table of Contents

List of Figures

List of Tables

VEHICLE MINIMIZATION FOR THE MULTIMODAL PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

I. Introduction

The General Problem

The algorithm developed here addresses a special case of the Vehicle Routing Problem (VRP). The VRP considered is multimodal with time constraints. It is also subject to capacity constraints, both on individual vehicles and on offloading at each node. This makes the problem broadly equivalent to the general formulation of the M++RP, or multimodal multicapacitated vehicle routing problem. However, the objective function used is based upon the total numbers of vehicles of each type used. Consequently, the problem can also be considered to be a special case of the Fleet Size and Mix Problem (FSMP).

The primary issue in solving any VRP is computational complexity. This is even more true when addressing the M++RP, or the FSMP. The computational complexity is exacerbated by the size of the problem and by the number of options available. It is not reduced by the constraints. In fact, the constraints may increase the computational complexity of the problem. The constraints may cause this issue by creating interference between the shipments in their processing at specific locations or on specific vehicles.

The FSMP has three specific issues related to computational complexity, which substantially expand the problem. Firstly, the FSMP does not directly allow for tradeoff-costing, either between particular days or between vehicle types. We concern ourselves with the total number of vehicles used and not with the vehicles used on any particular day. Because of this, vehicles can be considered ‘free’ with respect to the objective function except if their assignment would cause more vehicles to be required. It is not always possible to infer directly how the assignment of one vehicle might affect the assignment of later vehicles. Because of this, the costs of assigning vehicles are generally unavailable, directly, until the latter parts of the assignment.

Secondly, in the FSMP, we do not know the number of vehicles available for any given mode. This makes capacitating flow on any given mode difficult. It also makes it difficult to determine the limitations on the number of shipments any given mode is capable of carrying. Without knowing what a certain mode can carry, it is difficult to determine what the other modes must carry.

Thirdly, the FSMP generally requires multiple iterations to solve. We are seeking the feasible solution using the fewest vehicles. The intuitive approach to this solution is to determine a number of vehicles which is obviously sufficient. We could then reduce the number of vehicles, checking to ensure feasibility with each reduction. This is functional because it allows the determination of each mode’s capacity interactively with the other modes. All methods of solution must somehow take this process into account.

Generally, this means correcting an original solution, which of course requires many solutions of the VRP for a single FSMP solution.

Algorithm Overview

The algorithm developed in this thesis is decompositional. A diagram of the algorithm can be seen in Figure 1. It handles the TPFDD by splitting it into two data-components. The first data component is the network, which is processed by Dijkstra's algorithm. Dijkstra's returns a simplified version of this network. The second data component is the shipping requirements. The shipping requirements and network are then used as inputs for the second part of the algorithm. This part of the algorithm is the iterative scheduling heuristic. It efficiently allocates the shipments included in the shipping requirements to vehicles associated with paths on the simplified network. This assignment is made so as not to violate any of the constraints associated with the network, vehicles, or shipments.

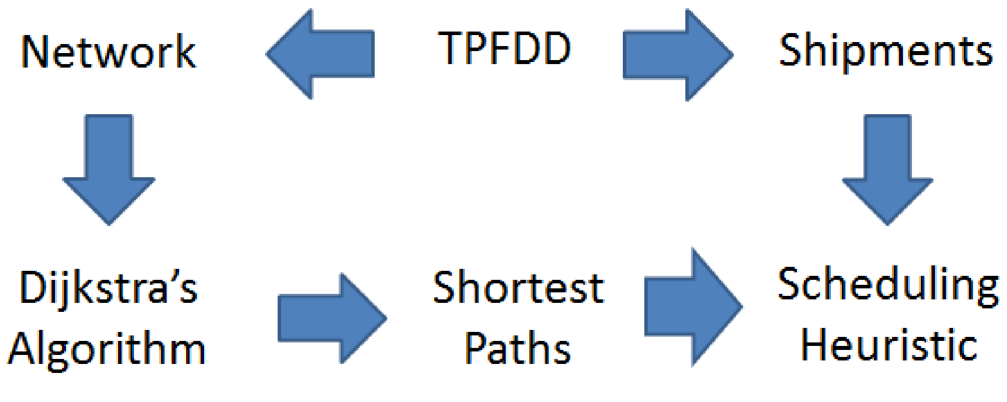

Figure 1. Overview of Algorithm

Dijkstra's algorithm is well-researched, very fast, and very reliable. In this instance it has only a small, conventional role. It seeks out the shortest path from each node included in the original network to each other node included in the original network. If we assume that the shortest paths are the best paths to use, we may use these paths in place of a full routing algorithm. While the paths are unlikely to be ideal, they approximate the ideal.

The second component of the algorithm is the iterative scheduling heuristic mentioned above. It iterates once for each mode. First, it determines the cheapest mode which is capable of carrying each shipment. At this stage, the algorithm assumes that the shipment is immediately loaded onto the mode on arrival, and when delivered, unloading capacity is immediately available. After this stage, the algorithm determines whether the number of shipments assigned to each mode is too great to be handled by the unloading capacity of that mode. Any excess shipments are moved to a higher-cost mode.

Finally, the number of vehicles required to carry any shipments assigned to the most expensive mode is determined. Then, any excess capacity is used to deliver shipments which were previously assigned to a cheaper mode. Any shipments which are assigned to the most expensive mode are eliminated from the overall shipment list. The process then repeats, disincluding the newly minimized mode. In the next iteration, the reduced shipment list is used.

Scope of Research

The purpose of this research is to develop an algorithm capable of solving a large instance of the multi-modal FSMP. The algorithm will emphasize deadlines and arrival dates as priority constraints in its solution. It will minimize vehicles in order of cost, ensuring that the most expensive vehicles receive priority. Each shipment will be shipped entirely on a single mode. Real-world aspects of the problem which are included in this solution include vehicle and offloading capacity and variation in path lengths.

Issues, Needs, and Limitations

The research is limited by its inability to model the real world with precision. Many constraints are applicable to the real-world problem but beyond the scope of the algorithm. This is generally due to added computational complexity. As a consequence, the results given by the algorithm can only provide a guide to the number of vehicles ultimately required for any given TPFDD and network.

A major limitation of the system as it stands is the removal of constraints regarding which modes can carry certain shipments. Certain shipments, according to a TPFDD, are locked into a particular transportation type. This algorithm does not allow for such limitations, but instead assumes that all shipments can, at least theoretically, be carried by any mode.

Another major limitation is the lack of multimodal solutions that is apparent in the algorithm. Due to simplifying steps taken early in processing, the algorithm cannot address the possibility of efficient or effective multimodal solutions. However, it is likely that such solutions can be generated by slight modifications to the final solution set, as with a genetic algorithm or Tabu search.

Research Organization

This chapter describes the general problem and the algorithm proposed in this research in general terms. It then continues with a discussion of the scope of the research itself, as well as the limitations of the algorithm and the research. Chapter II describes the background of the Vehicle Routing Problem (VRP), emphasizing the Dial-A-Ride Problem (DARP) and the Multimodal Multicapacitated Routing Problem (M++RP) as special cases. It continues with a discussion of solution techniques for the DARP. Afterward, it discusses the Q-machine scheduling problem, and techniques which are used in solving Q-machine scheduling problems. Finally, it discusses the pragmatic instance studied in this research. Chapter III presents the algorithm in detail. It begins with an overview of the algorithm. It continues by discussing Dijkstra's algorithm, and its role in the algorithm. Then it discusses the scheduling heuristic. As part of its discussion of the scheduling heuristic, it first covers the deadline and infrastructure assignment steps, and then covers the vehicle assignment step, ending with the correction step. Chapter IV presents four data sets which are used to test the algorithm. The first data set discussed is a simple data set used as a demonstration. The second data set examined extends the first case to multiple modes. The third data set is a modification of the second, intended to highlight the algorithm's reactions to infeasibility. The fourth data set is a stress test of 500 shipments, to show the speed of the algorithm. Finally, Chapter V provides an overview of the efficacy and drawbacks of the algorithm, the conclusions of the research, and suggestions for further research.

II. Literature Review

Chapter Overview

The problem being reviewed in this research is a special case of the Dial-A-Ride problem, itself a special case of the Vehicle Routing Problem, the M++RP (Moccia et al, 2008) The Dial-A-Ride problem is a subset of the general pick-up and delivery problem, as defined in Savelsbergh and Sol's "The General Pick-up and Delivery Problem" (Savelsbergh et al, 2005) but the pickup and delivery problem is itself a refinement of the "Truck Dispatching Problem" originally proposed by Dantzig and Ramser in their eponymous paper. (Dantzig, 1959)

The Vehicle Routing Problem has received a great deal of attention over the years, but the particular refinement being dealt with in this paper is substantially less studied. In particular, the M++RP deals with multi-modal, multi-time constraints, and multiple capacity constraints, in addition to the constraints more usually associated with the VRP, but without depots. (Moccia, 2008: 2)

The problem can, however, also be approached as a special case of the sequential machine scheduling problem for non-homogenous machines, or the Q-machine scheduling problem. While the problem can be viewed as such, the more interesting applications of Q-machine techniques for this problem relate to the use of the Q-machine methods for solving the 'scheduling' component of a decomposed problem, and so that is where our study will focus.

This chapter will first review the Vehicle Routing problem solution techniques. It emphasizes the state of exact algorithmic solutions, and the decomposition and heuristic mixed methods for achieving large-scale near-optimal solutions. It also discusses the difficulty of achieving a reasonably accurate solution in a short time-scale on a problem with such great computational complexity. Then we review the techniques used in Q-machine scheduling as approaches to the solution of the decomposed problem.

Vehicle Routing Problem Overview

The vehicle routing problem is the problem consisting of finding an optimal route for either one or multiple vehicles between multiple locations, each of which will generally place a load on the vehicles, to be transported to a second location. This second location may be the depot of the truck, in simpler problems, but is often a delivery location. In this case, the problem becomes the Vehicle Pick-up and Delivery problem; more specifically, the problem may be constrained to require that the pick-ups and deliveries occur according to a certain schedule, in which case the problem becomes the Vehicle Pick-up and Delivery Problem with Time Windows.

Dial-A-Ride Problem Summary

The most studied problem class which closely resembles the one discussed in this paper is known generally as the ‘Dial-A-Ride’ Problem. The Dial-A-Ride problem is a special case of the Vehicle Pick-Up and Delivery Problem with Time Windows, with vehicles operating from and returning to an established depot. The problem is subject to vehicle capacity constraints and constraints on the maximum amount of time a customer may ride in the vehicle.

The Dial-A-Ride problem differs from the studied problem in several particulars. The first is that the Dial-A-Ride problem does not generally have to deal with infrastructure constraints on loading or offloading of shipments. This allows the problem to be simplified in significant ways. First, a node occupied by a vehicle's unloading is unusable to other vehicles from other paths. The vehicles, however, may interfere with one another if the capacity constraints on the arcs are used to create that effect. The second is that the Dial-A-Ride problem generally deals with homogeneous vehicles, rather than the multimodal approach required in dealing with the studied problem. This creates additional computational complexity, for two reasons. The first is that the various vehicles can be traded off, one against the other, providing another aspect of complexity, rather than simply requiring the addition of more homogeneous vehicles (as in the Dial-A-Ride problem). The second constraint is that the path from any given node to any other node is unique in the Dial-A-Ride problem, as generally understood, rather than having different distances and speeds for different modes. Finally, perhaps the biggest difference between the Dial-A-Ride problem and the problem studied here is the problem of scale. The Dial-A-Ride problem generally deals in vehicles which are each capable of handling multiple loads, whereas the problem studied here generally deals in loads which will require multiple vehicles. Hence, in the Dial-A-Ride problem, the core issue is ensuring that the vehicles waste as little travel time as possible in getting as many loads as possible to as many locations as possible. For the studied problem, the emphasis must be on ensuring that the correct vehicles travel to the correct locations at the correct times, so as to prevent conflicts, and most importantly ship them efficiently and cheaply. Finally, the Dial-A-Ride problem is often solved for a single vehicle, rather than for multiple vehicles. While the Dial-A-Ride problem can be extended easily from a single-vehicle technique to a multi-vehicle solution under many circumstances, the particulars of the M++RP make it ineffective to extend from a single vehicle solution to multi-vehicles, especially since the particular problem being studied has as one of its primary objective function the use of minimum numbers of vehicles of each type.

Dial-A-Ride Problem Solution Techniques

The Dial-A-Ride problem is computationally complex, but also very fragile. The number of variables involved means that the cost of accurate solutions to large-scale problems is often prohibitively high, and instead heuristic models must be used. Nonetheless, exact solutions can and have been found for smaller problems. There has also been significant research into the extension of exact solution techniques for problems after decomposition or alteration. However, the most interesting part of these techniques, for our purposes, is the development of decomposition and simplifying techniques to be used in conjunction with heuristics. For the Dial-A-Ride problem, it is often possible to simplify the problem to the point where an exact solution to the problem becomes feasible, even if the problem loses some fidelity in the process; for the M++RP problem, and more specifically, for the pragmatic instance of the M++RP problem being studied in this paper, simple decomposition will not result in an exactly solvable set of problems.

In 2004, Lu and Dessouky demonstrated a method for efficient generation of exact solutions to the Dial-A-Ride problem. The method was reliant upon an integer programming formulation of the problem, which was then solved using a simple branch and bound technique. It solved a problem consisting of 5 vehicles and 17 customers in less than three hours. This demonstrates the complexity of the problem, since using only 5 vehicles and 17 customers generated that level of computational demand. The advantage of Lu and Dessouky's innovation was that it added a level of softness to calculations regarding time and capacity constraints; however, even with these significant changes to the fundamental paradigm, the algorithm produced a relatively time-costly solution to a relatively small problem. (Lu et al, 2004)

Psaraftis (Psaraftis et al, 1980, 1983) demonstrated an exact algorithm for the solution of the Dial-A-Ride problem in Transportation Science, dealing with multiple vehicles. His technique provides an exact solution, using a dynamic programming algorithm, which efficiently and effectively calculated the best method for dispatching the vehicles, including route and schedule. Originally, Psaraftis developed the technique for a fairly simple variant, involving only one vehicle, but it was eventually extended to fairly complicated multi-vehicle variants, including time constraints. The downside of Psaraftis' approach is that it only optimizes with respect to total distance travelled. While total distance travelled is of a certain commercial interest, it is effectively irrelevant to our particular problem because of the scaling issue. In our problem, distance travelled is a concern secondary to our primary goal- as we know, ultimately, that our vehicles must travel from pickup to delivery, and then to pickup, rather than the interchanging sequence possible in the generic Dial-A-Ride problem. Equally, Psaraftis' solution does not address the problem of total vehicle number, which is what our algorithm is ultimately designed to address.

More useful to us in this instance is the generation of large-scale solutions to the Dial-A-Ride problem, which generally involves a heuristic approach. In most cases, the approach consists of a simplifying step, followed by an algorithm which approximates solutions to the reduced problem. In simpler cases, the problem is simply reduced directly using an analytical approach as in the generation of lower and upper bounds, and then solved exactly, or very closely. In large scale cases, the problem is decomposed and then approached with a heuristic technique, which provides a lower-quality but equally lower-cost solution to the problem, and is often the only feasible approach to such a problem.

Baldacci et al (Baldacci, 2011) begin by generating a specialized integer formulation of the problem, and then the dual of that form. They then use two heuristics in conjunction to achieve a near-optimal resolution of the dual, which in turn they use to determine which paths meet certain lower-bound and upper-bound criteria. They then remove all paths which are outside these bounds, and solve the reduced problem using an integer programming technique, or if the problem remains too large, attempt to resolve the size disparity using branch-and-bound techniques.

Sexton et al (Sexton, 1985) relied on Bender's decomposition, separating the problem into a 'routing' component and a 'scheduling' component, and then solving with a heuristic. This technique is very efficient for the resolution of the Dial-a-Ride problem, because the paths are effectively independent of scheduling. If one can determine which paths are most efficient, then the problem should nearly always solve optimally subject to those paths, which allows for a drastic reduction in the complexity of the problem.

While these decompositional techniques are effective in resolving some of the computational complexity of the problem, the decomposed problems remain very complex. Even after decomposition of the problem, we are left with a routing component equivalent to solving the shortest path problem for each of the customers, and a scheduling problem for parallel machines. While this technique is reasonable for resolving a problem involving only a few nodes, arcs, customers, and vehicles, handling the problem becomes substantially more difficult at larger scales.

An approach to multi-modality for a flexible number of vehicles was developed by Moccia et al (Moccia, 2008), and focused upon the use of column generation heuristics. In this case, the formulation of the problem used ‘virtual networks’ to represent multimodal shipment transfers, developing false links with associated cost functions and time costs to represent the price of transferring from one mode to another at a given linkage. This methodology results in a reasonable solution for relatively large variants of the VRP. However, the algorithm used in the paper could only handle a relatively small system, though with great fidelity.

Dial-A-Ride Discussion Summary

Solutions to the Dial-A-Ride problem are very rarely exact, depending instead on heuristic algorithms, often combined with decomposition, to solve even relatively simple problems. This is partly due to the limitations imposed by integer programming formulation, which the majority of the techniques use as a beginning for their solution. Any integer programming formulation must address tens of thousands, or even hundreds of thousands of variables, addressing which path, if any, each vehicle must be on at which hour of which day, carrying what load. By extension, almost any solution to a large-scale Dial-A-Ride problem relies on a simplifying step, followed by powerful heuristics- and even so, will generally provide only a relatively slow solution to a relatively small problem.

For more details on the history of methodologies for studying the Vehicle Routing Problem, readers are recommended to *Fifty Years of Vehicle Routing* (Laporte, 2009) in the 43rd issue of Transportation Science.

Q-Machine Scheduling Summary

One approach to the M++RP problem is to reduce it to a scheduling component and a routing component. Once the shortest routes have been determined, the problem can then be handled as a scheduling problem, treating each of the modes for each path as a machine, with the vehicles treated as a global resource shared between the machines.

The transformation of the problem to a Q-machine scheduling problem reduces the complexity of scheduling significantly, but we are left with a computationally demanding problem nonetheless. At this stage, integer formulation of the resulting problem becomes more feasible and extensible to very small variants of the problem (Wagner, 1959) but the establishment of a more effective heuristic technique remains necessary for moderate to large scale scheduling problems. (Verma, 1999) In many cases, the most efficient method remains a scheduling 'rule,' modified as necessary by evolutionary algorithmic techniques to improve upon the initial high quality solution. Because of the issues of interference between various shipments, even a minor shift in the location of a single shipment can have major cascading effects on the efficacy of the solution as a whole. This is particularly true as the chosen metric, number of vehicles used, is dependent upon peak usage across the various vehicles, not upon the total usage of each vehicle type.

Q-Machine Scheduling Heuristics

The simplified form of the M++RP that we are solving in this instance is equivalent to solving $Q_m|s_{ijk}|\sum w_jT_j$, or the parallel machine scheduling problem with machines with non-equal speeds, which are not dependent on the specific job, with setup times, in order to minimize tardiness, and then modifying that schedule in order to minimize m, while holding the previous objective value static. This variant of the problem has seen significant research because of its industrial significance, and consequently, many algorithms have been developed and applied to the problem. However, extending an exact solution to large instances of the problem remains elusive.

Most solutions to the $Q_m$formulation for large problems rely upon a prioritization heuristic, but unfortunately no single index appropriately addresses $\sum W_jT_j$, and even if it did, the flexible nature of the number of machines means that while we could solve for weighted tardiness, we would not be able to prioritize reduction of machines; priority rules by definition assign a job to the first free machine, rather than attempting to reduce total machine numbers.

For $\sum W_jT_j$ specifically, priority rules are difficult to implement because of the complex nature of allocation. No specific variable, ratio, or difference can provide an efficient and effective index in all instances. Instead, the Apparent Tardiness Cost with Setups (ATCS) prioritization rule was developed, as a combination of all of the factors which might cause a particular job to be the priority job for a particular freed machine, weighted dependent on the particular characteristics of the machines and jobs.

The ATCS is one heuristic which has been developed to handle $Q_m | s_{ijk} | \sum w_j T_j$. The ATCS calculates an index based upon the processing time, setup time, objective weights, due date tightness and range factors, and the severity of the setup time. When a job is completed, the job with the next highest index is assigned. The Apparent Tardiness Cost with Setups is very efficient at handling large scale problems, and is equally very effective at generating an optimal or near-optimal solution. However, the ATCS does not effectively handle the in-parallel nature of the infrastructure constraints which are to be dealt with in the current problem, simply because those constraints are not factored into its system, and requires as part of its algorithmic structure the existence of a defined number of vehicles. Regardless, the ATCS is a very efficient approach to the large scale problems being handled in this instance for minimization of weighted tardiness. It is worth noting that in the seminal paper on the topic, Lee (Lee et al, 1997) used a corrective simulated annealing technique to improve on the value of his final solution, relying on the ATCS rule only to generate a feasible high quality initial solution.

Beyond the constructive algorithms designed to generate a feasible and near-optimal solution, we find refining algorithms designed to improve on an existing schedule. These techniques generally apply a local search heuristic, moving from one good solution to similar solutions stepwise. Two of the most commonly used heuristics in this role are Simulated Annealing and Tabu Search, each of which searches locally for improvements to the currently generated schedule.

Simulated Annealing techniques depend upon a large number of solutions, which are randomly traded for other ‘nearby’ solutions. Better solutions are generally preferred, and as the algorithm progresses, the preference for better solutions increases, until the algorithm is simply stepping to the local optimum. Similarly, Tabu search allows for an algorithm to pass into infeasible territory, if the objective function can be improved by doing so, by providing a penalty function associated with the infeasibility. As the algorithm progresses, it increases the penalty to achieve an effective hard feasibility.

Q-Machine Scheduling Summary

The solutions to Q-Machine scheduling expose us to the idea of prioritization rules which allow for the solution of the problem for particular objective functions. These methods are not effective for solution subject to the specific criteria of vehicle minimization, but they provide a starting place for the development of our own rules and solution index.

Simulated annealing and the Tabu search show us the next potential stage of the development of the research, which is to develop a refining algorithm. Tabu search, simulated annealing or evolutionary algorithms can be used to refine the solution into a specific high quality solution. The difficulty in implementing such a solution lies in the complexity of handling hundreds or thousands of large scale solutions to the problem. Without those, the refinement the heuristics can provide is minimal; with them, the algorithm becomes cumbersome.

Pragmatic Instance Summary

In this research, the particular problem we are studying is the M++RP problem, at large scales. The particular instance of the problem which is being discussed which we are using as a pragmatic instance of our general problem is a Time Phased Force Deployment Data, or TPFDD. A TPFDD consists of a large number of transportation requirements, from a number of sources to a number of sinks, across a defined network. The development of transportation requirements and vehicle numbers required to move them is an interactive multi-stage process, as the number of vehicles themselves necessitate infrastructure and movement capacity at the vehicle level. According to Clausewitz' *Principles of War*, "The provisioning of troops, no matter how it is done, whether through storehouses or requisitions, always presents such difficulty that it must have a decisive influence on the choice of operations."

As a consequence, it is of particular importance to be able to quickly generate reasonable estimates as to the number of vehicles of various types required to execute a TPFDD, as generating these estimates will most likely be required multiple times, in a feedback process with both analysts and decision-makers. However, a TPFDD is remarkably large; as many as ten thousand transportation requirements (customers), across a network of as many as several hundred nodes, with multiple modalities, over the course of weeks or even months, subject to constraints on earliest and latest arrival, as well as to constraints on infrastructure for offloading that will be available, and potentially to many other over-riding constraints which are beyond the scope of this model.

It is a priority in all strategic situations to ensure that all deadlines and earliest arrival dates are met, subject to feasibility; the number of vehicles used is secondary to the accomplishment of the purpose behind the TPFDD, which may rely on any particular requirement. For this reason, the algorithm generated here must prioritize that all deadlines are met and only as a secondary concern handle the vehicle minimization techniques.

With that caveat, the solution of a vehicle minimization problem requires a complete solution for the problem including vehicle allocations and paths, because of the interaction of infrastructure capacity requirements. Because of this, and because any given solution will tend to depend on the number of vehicles available, the problem must be solved multiple times during any particular attempt to minimize the vehicle numbers. At the very least it must be solved once for each mode. This places an even higher priority upon high processing speed than was already necessitated by the size of the problem and the requirement for interactive feedback.

In the particular problem being studied it is noteworthy that the shipments will nearly always require multiple vehicles to carry; this allows for certain simplifications and changes of emphasis in the details of our algorithm. It is also worth noting that since the particular scope of the TPFDD is in-theater, in our application we are unlikely to find a solution that requires transshipment from one mode to another; instead, despite the multi-modal nature of the problem, we may with reasonable safety confine ourselves to the use of single modes for the duration of the trip, assuming the cost and availability of transshipment to be prohibitive.

Chapter Review

Our final summary concludes that the particular computational complexity of the large scale M++RP requires an approach which is computationally simple, as in the Q-Machine scheduling priority rules, applied to a simplified problem generated according to the decompositional rules used for smaller Vehicle Routing Problems. With the combination of these two techniques, we can drastically reduce the calculation time required for the generation of a feasible solution, without sacrificing unduly the optimality of our solution. The interactivity of our pragmatic instance specifically encourages this, as the solutions are intended as springboards for analytical thought, rather than implementable final answers.

With this sort of rough-cut approach to a problem of this computational complexity, the emphasis must be placed upon reducing the processing time required to handle the problem. Without careful management of processing time, we run the risk of an impractical or impossible technique, which will fail to generate the timely, effective solutions required.

For this reason, our ultimate implementation relies upon a series of priority rules, applied in careful order to the shipments, and solved in a specific order in order to preserve feasibility, while minimizing vehicle number requirements.

III. Methodology

Broad Analysis of the Algorithm

At the highest level, the algorithm consists of three major steps. First, the incoming data is separated into a network component and a shipping requirements component. Second, the network component is processed using Dijkstra’s algorithm, to create a network of shortest paths. Finally, the scheduling heuristic assigns the shipping requirements to the simplified network at need. This organizational hierarchy will serve as the structure for this chapter, as we follow the flow of data processing throughout the algorithm.

Data Inputs

The algorithm requires three different major data components. The first of these components is the network itself. The network is composed of a series of nodes, with associated distances between them, and a value for the daily unloading capacity of the nodes in the units which are later used for shipment weight. Each of these distances and unloading capacities must be defined for each mode. In the case of a node-node pairing which cannot be travelled by a specific mode, it is possible to assign a ‘big M’ value for the transportation distance in order to force the shipment onto a higher-cost, but feasible, mode of transport. However, doing so can only cause the algorithm to transfer the shipments upward in cost.

The second component of data required is vehicle information. The algorithm requires data on vehicle speed, capacity, and the number of modes. This must parallel the number of parallel modal networks provided. These data are used throughout the algorithm. Speed, particularly, is used in all three major components of the scheduling heuristic, either directly or indirectly.

The third component of data is shipping requirements. Shipping requirements are stored as a series of lists. Instead of directly manipulating the data associated with the shipment, the algorithm uses the number of the shipment as a serial. Moving only integer values significantly reduces the time required to sort and generate lists.

Dijkstra's Algorithm

Dijkstra's algorithm is used in place of a more complicated routing solver in order to approximate the ideal routes for vehicles. The shortest paths generated by Dijkstra's are good approximations if the vehicles are generally required to return to their depot after delivery to only one site. If this is held to be so, the routing problem becomes generally the problem of travelling from point A to point B to point A as efficiently as possible. This is equivalent to the shortest path problem. The implementation of Dijkstra's algorithm in this research is illustrated below in Figure 2.

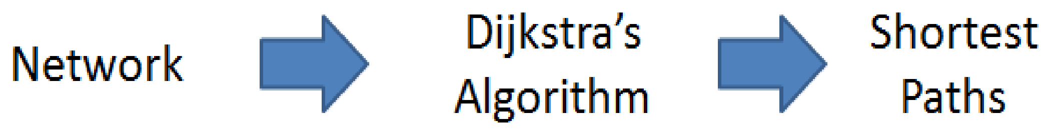

Figure 2. Dijkstra's Algorithm

Dijkstra's algorithm is a very efficient algorithm for solving the shortest path problem, provided that the distance desired is from each node in a network to each other node in the same network. It works by expanding upon paths of known distances and tracking the shortest path discovered to each node. At each step, it advances to the next nearest node to the origin node. It records any nodes for which the shortest known path is longer than the distance to the current node from the origin node, plus the distance from the current node to the observed node. It then corrects their distances down to the newly discovered shortest path. Finally, it advances to the node which is the next closest to the origin node, after the currently selected node.

Scheduling Heuristic Overview

The scheduling heuristic used in this algorithm is ultimately the core of the entire procedure. Dijkstra's algorithm can be viewed as a pre-processing stage that puts the input into a form conducive to the use of the scheduling heuristic. The scheduling heuristic bears special attention, especially as it comprises the majority of the complexity of the algorithm as well as the key part of its function.

The scheduling heuristic, as shown in Figure 3, has four key parts; these are: Deadline Assignment, Infrastructure Assignment, Vehicle Assignment, and the Correction Step. Deadline Assignment and Infrastructure Assignment can be viewed as pre-processing steps, Vehicle Assignment as the core step, and the Correction Step as a post-processing method. However, each of these steps will be iterated once for each mode, as the overall heuristic determines the minimum number of vehicles required for only one mode at a time.

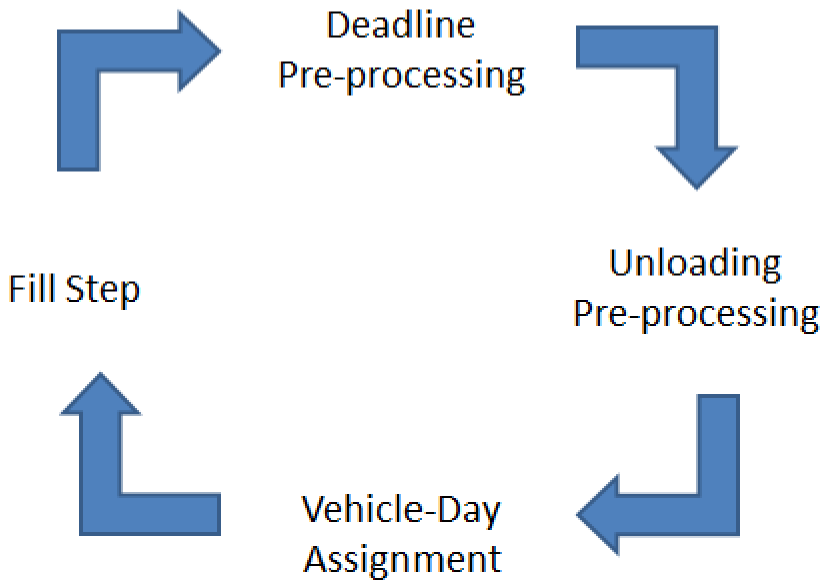

Figure 3. Scheduling Heuristic Overview

Scheduling Heuristic Inputs

The inputs for the scheduling heuristic have two sources. The first is Dijkstra's algorithm, mentioned above, which provides us with a simplified network of shortest paths for use in the calculation of distances throughout the heuristic. The second is the shipping requirements component of the original data, which is passed on in the form of a list of shipment numbers and a series of associated lists detailing arrival date, shipment size, and deadline, all accessible using the shipment number as a serial. The algorithm also acquires the vehicle data directly from the original listing.

Deadline Assignment

Deadline analysis is the simplest of the four stages of the scheduling heuristic, and the quickest. In deadline analysis, each of the shipments has a time-available value calculated, which is simply the difference between arrival date for the shipment and the deadline date. This is the amount of time that a shipment is available for shipping. We compare this value to the speed of each mode and the distance for that mode between the source and sink for the shipment, then, add the amount of time required to unload the shipment. A mode for which distance/speed plus unload time is greater than the time available certainly cannot carry a given shipment. As a consequence, we know that the shipment must be moved higher in cost- to a faster mode.

Deadline analysis serves two functions simultaneously. First, it ensures that shipments which would be required to run on a more expensive mode for reasons of available time are assigned upwards earlier. This saves the time of calculating that they must be pushed up during the more computationally intensive infrastructure assignment stage. Second, it ensures that these shipments cannot cause other shipments to be forced upwards during the infrastructure stage.

When a shipment’s cheapest potentially feasible mode has been determined by the deadline function, it is assigned to a list associated with that mode. There is a list for each mode at the end of the deadline stage and each shipment will be in one, and only one, of those lists. These lists form the input for the Infrastructure Assignment stage of the algorithm.

Infrastructure Assignment

Infrastructure Assignment can be viewed as another preprocessing stage of the algorithm. However, it is also fair to consider the Infrastructure Assignment stage as the stage of the algorithm wherein the unloading constraints are taken into consideration. While unloading is considered at the Vehicle Assignment stage as well, it is at this stage that it is most likely to cause a shipment to be moved or bumped from a mode, as opposed to simply forcing rescheduling. In other words, this is the stage where overall capacity of infrastructure unloading is taken into account.

This is achieved using a 2-dimensional array. Because we handle each mode separately, it is not necessary to maintain the full node-mode-day pairing for tracking unloading. Instead, we simply track the node-day pairing for the mode which is currently being analyzed.

The algorithm starts from the earliest arrival date, and begins to check through the list of shipments assigned to the particular mode. As it iterates through the shipments, if it finds any shipment with the arrival date it is currently searching for, it attempts to assign them immediately to the mode. If it fails, it adds them to the list for the next most expensive mode. If there is no more expensive mode, the shipment is retained at this mode. After processing through the list once, it increases the arrival date by one and processes through again. This is repeated until all shipments have been processed through the system.

Processing based upon earliest arrival date is known as the EAD priority rule. This rule has several advantages. Primarily, it ensures that the infrastructure begins work as early as possible. That is to say that since no shipment can arrive prior to the shipments with the earliest arrival date, if they are the first shipments assigned, we can guarantee minimal lead-time, which helps in reducing wasted processing time.

EAD is approximately equivalent to the First Come First Served (FCFS) rule, which is intuitively a very efficient means of ensuring that the infrastructure is efficiently used. The primary failing of FCFS and EAD is relative to rules such as Shortest Processing Time or Weighted Shortest Processing Time. EAD is efficient at ensuring the maximum possible tonnage is carried, but does not account for weighting across tonnages. Fortunately, in our case, it is assumed that all shipments have equally inviolate priority.

The process of assignment for infrastructure is a relatively simple one. Each shipment is taken in order, and the algorithm searches the array to attempt to find space to unload it. At this point, we do not concern ourselves with vehicles. However, we do add the constraint that no shipment can be unloaded before its arrival time plus time of travel to the unloading point.

In order to search the array for the appropriate amount of time, we first calculate the time required for unloading. This is simply the size of the shipment divided by unloading capacity. We then find our start point, which is the arrival time for the shipment plus the travel time required on the mode in use. Finally, we iterate from this point to the deadline for the shipment, summing all free time we find.

The array used to track the amount of free infrastructure capacity is made up of the amount of capacity free on any given day. Each value is between 0 and 1. If the value is 0, the day is completely free. If the value is 1, the day is completely full. Any value other than these two represents a partially used day. The algorithm adds the remaining portion of the day for each day between the start time and the deadline, except the first. For that day, it adds the remainder only if the already allocated portion of the day is larger than travel time. This prevents the shipment from being treated as unloading while it is still in travel.

If the algorithm finds sufficient space for the unloading of the shipment, then the shipment is added to the output list for this mode. If it does not, then it is added to the output list for the next most expensive mode. It is not necessary at this stage for the loading to be contiguous, as the specifics of assignment are handled at the vehicle assignment stage.

Vehicle Assignment

The vehicle assignment algorithm is the core of the scheduling heuristic. It receives a list of shipments which must be assigned to the most expensive mode from the infrastructure assignment component, and it converts that list into both a detailed schedule and a requirement in terms of number of vehicles. Because it is so essential, and because it is complex, it merits a more detailed look than either the deadline or infrastructure components of the heuristic.

The vehicle assignment algorithm uses the same method of selection for shipments as the infrastructure method. It chooses them based on earliest arrival date, tracking down through the assigned list, iterating each arrival date in turn.

Once the shipment has been chosen, the algorithm first determines the number of vehicles necessary to carry the shipment. This is the size of the shipment divided by the capacity of the vehicle, rounded upwards. The algorithm then begins the search for appropriate vehicle and unloading space for the shipment.

The first step in this process is identifying a free space on a vehicle. Much as in the infrastructure array, we use an array to track the usage of the vehicles. Unlike in the infrastructure array, however, we must seek to gain continuous use of the vehicle for the full duration of the trip. So rather than simply beginning at our starting point and proceeding to deadline, summing the free space, we use a rather more complicated summation process. We begin at the arrival day for the shipment and iterate through the chosen vehicle's days. If we find a day that is empty, we add 1 to our currently found free time. If we find a day that is not empty, we add the remainder of its capacity to our currently found free time. If, after adding the new capacity, the free time found is greater than the amount needed, we mark our original start time as an appropriate start time for the shipment-component and proceed to the infrastructure correction step. If not, we set our total free time equal to the remainder, and set the current day as our start time.

If we fail to identify a free spot large enough to carry the shipment and our vehicle isn't one that was generated just for this shipment, then we generate a new vehicle to carry the shipment. If the vehicle was generated just for this shipment, then we ignore the deadline limitation and assign the shipment as late, if necessary.

If, however, we identify a free spot in the vehicles where the shipment could be carried, we must now confirm that there is infrastructure available to unload the shipment in the appropriate place. This process is identical to the process for finding free space on a vehicle, except that new infrastructure cannot be generated. If we find ourselves pushed past the deadline on infrastructure, we instead simply must assign past the deadline.

If the start time found by the vehicle-search is confirmed by the unload-search, then we may add it to our list and begin searching for vehicle and unload space for the next shipment-component. However, if it is not, we find the next start time available among the vehicles, starting at the one suggested by our unload-search. If the start time required correction, we repeat the process until the shipment is assigned.

If at any time we are forced to use a new vehicle, we track the number of this vehicle. The last vehicle we are forced to generate is the minimum number of vehicles required to service this set of requirements.

Finally, when we have found appropriate start times for all the components of a shipment, we allocate the shipment and fill the capacity in the unload and vehicle arrays. It is at this point that we remove the shipment from the shipment requirements array, to represent that it has been assigned.

Correction Step

The correction step is the final process in the scheduling heuristic. In this stage, the algorithm uses the same search procedures used in the vehicle assignment stage, iterating through all unassigned shipments in EAD order. However, if it fails to find capacity, it simply moves on to the next shipment, rather than generating a new vehicle.

It treats the number of vehicles generated by the vehicle assignment step as a capacity on the amount of flow the vehicles are capable of handling. However, if a shipment can be assigned, it is assigned and deleted from the shipment list.

IV. Results and Analysis

Simple Test Case

In order to examine the capabilities of the algorithm, a series of test cases were created. The test cases were deliberately chosen for ease of solution, in order to make comparison against an intuitive or obvious perfect solution, simple for the reader. In the first case, our study case is hyper-simplified and consists of only 5 shipments, each with a three-day gap between arrival and delivery, arriving at the same source node, one per day, over a five day period, all of which are destined for the same sink node.

For simplicity's sake, this example deals in only one mode and the sizes of the five shipments are equal to the capacity of the vehicles, resulting in exactly one shipment being carried by each vehicle per trip. The vehicle was given a speed of 100, and the distance between nodes 1 and 2 was set to 100; also, the unloading capacity for the sink node was set to 100, to simplify displaying the outcome. Table 1 summarizes the shipping requirements for the system.

Table 1. Simple Test Case Data Input

Shipment	Size	Source	Sink	Arrival	Deadline
S1	100	1	2	1	4
S2	100	1	2	2	5
S3	100	1	2	3	6
S4	100	1	2	4	7
S5	100	1	2	5	8

The algorithm returned the following, displayed in Table 2, as a feasible resolution of the system, determining that the number of vehicles required for such a solution was three.

Table 2. Simple Test Case Results Output

Shipment	Start Time	Arrival Time	Vehicle Number	Ahead of Deadline
S1	1	4	1	Yes
S2	2	5	2	Yes
S3	3	6	3	Yes
S4	4	7	1	Yes
S5	5	8	2	Yes

Examining the solution shown in Table 2, it is possible to chart the assignments which each of the three vehicles had for the duration of the transportation solution; for clarity, the schedule produced is shown below in Table 3.

Table 3. Simple Test Case Vehicle Schedule

Vehicle	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
1	S1 delivery	S1 unload	S1 return	S4 delivery	S4 unload	S4 return	Idle
2	Idle	S2 delivery	S2 unload	S2 return	S5 delivery	S5 unload	S5 return
3	Idle	Idle	S3 delivery	S3 unload	S3 return	Idle	Idle

Modified Simple Case

The algorithm, then, is capable of achieving an intuitive result on a small scale. Given that this test case is equivalent to the case where all shipments can be carried on a single cheapest mode, that data shall not be repeated here. However, a secondary component of the algorithm which bears examination is its capacity to compensate for a shipment being forced onto a high cost mode, by utilizing the idle capacity of that mode to the greatest extent possible. As a consequence, our second test case, the modified simple case, will change the base case in two ways. First, it will add a second mode, considered more expensive than the first, which travels at a rate of 200 units per day, with identical independent unloading capacity to the first mode. Second, it will change the deadline on the first shipment to 2. The input for this case is shown in Table 4.

Because the first shipment has a deadline of 2, it is impossible for it to be delivered by the deadline using the first or second modes, and so it will be forced to travel on the second mode in order to minimize the violation of the deadline (guaranteeing a minimum number of vehicles for the second mode of at least one). However, the rest of the vehicle's time is not accounted for, and so we must allocate as many shipments as possible to the idle time on that vehicle in order to minimize the number of lower cost vehicles used.

In this instance, the utilization of the mode allows for two additional shipments to be handled by the most expensive mode, on the same vehicle that is handling the first shipment, resulting in two fewer vehicles being required on the first mode.

Table 4. Modified Simple Case Data Input

Shipment	Size	Source	Sink	Deadline
S1	100	1	2	2
S2	100	1	2	5
S3	100	1	2	6
S4	100	1	2	7
S5	100	1	2	8

In this case, it is worth detailing the differences between the two modes, as well as their similarities, which can be seen in Table 5:

Table 5. Modified Simple Case Vehicle Data

Vehicle	Capacity	Unload Rate	Speed
1	100	100	100
2	100	100	200

The chart of our delivery times, travel times, and vehicle pairings is given below in Table 6, and below that, the chart of the vehicle time-assignments is given.

Table 6. Modified Simple Case Shipment Schedule

Shipment	Start Time	Arrival Time	Vehicle Number	Ahead of Deadline
S1	1	2.5	1 (Mode 2)	No
S2	3	4.5	1 (Mode 2)	Yes
S3	3	5	1 (Mode 1)	Yes
S4	5	6.5	1 (Mode 2)	Yes
S5	6	8	1 (Mode 1)	Yes

Because the second mode performs deliveries in one half-day, the pattern is that on the 'outgoing' day, the travel is completed outgoing, and half of the unloading is done, and on the next day, the remainder of the unloading is completed, and then the shipment is returned, as demonstrated by Table 7.

Table 7. Modified Simple Case Vehicle Schedule

Vehicle	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day7
1 (Mode 2)	S1 Out	S1 Return	S2 Out	S2 Return	S4 Out	S4 Return	Idle
1 (Mode 1)	Idle	Idle	S3 Out	S3 Deliver	S3 Return	S5 Out	S5 Deliver

This demonstrates the capacity of the algorithm to combine minimization techniques in order to reduce the impact of forced increases in the number of high-cost vehicles being deployed, by utilizing those new, largely idle, vehicles in order to decrease the number of vehicles required at lower tiers. Of course, in a more complex problem, resolutions will be substantially more complicated to come by, and in most cases, less efficient, and less obviously so. However, the general principle of efficient allocation still holds in more complex cases.

Multinodal, Multimodal Demonstration Case

The complexity of the algorithm, and its potency, rests upon its capacity to deal efficiently, and quickly, with problems that handle both multiple sources and sinks, and multiple modes, on large scale, but of course it is difficult to demonstrate efficiency in the large scale, because by-hand and intuitive solutions are hard to come by. Instead, we examine the efficacy of the algorithm using a smaller multimodal, multinodal pattern.

In this instance, our case involves three modes, over four nodes, each equidistant at 100 units from each other node, and each capable of unloading 100 units per day from each mode. For this problem, the speeds for our three vehicle types are 50, 100, and 200 units and the capacities are the same. We will process twelve shipments across the nodes, using nodes 1 and 2 solely as sources, and nodes 3 and 4 solely as sinks. The sum of all tonnage shipped is 2400 and the arrival times and deadlines are broadly separated, allowing for a powerful estimate of 2400/10 or 240 tons per day of shipping power being a floor on the number of tons per day of vehicle required to handle the shipping.

Of course, at this tier of processing, the outputs become significantly more complicated. As a consequence, we will simply list the vehicle number, node, start time, and mode for each shipment and shipment-component, as shown in Table 8.

Table 8. Multimodal Simple Case Data Input

Shipment No.	Source	Sink	Size	Arrival	Deadline
1	1	3	100	1	7
2	2	3	200	1	7
3	1	4	300	1	7
4	2	4	100	1	7
5	1	3	200	3	10
6	2	3	300	3	10
7	1	4	100	3	10
8	2	4	200	3	10
9	1	3	300	1	7
10	2	3	100	1	7
11	1	4	200	3	10
12	2	4	300	3	10

This list is not the original data output from the program, but has been sorted to highlight both the process behind the assignment of shipments to specific vehicles and a problem induced by the specific granularity of the shipments. Note that the shipments fit neatly together, as regards unloading. This is due to the algorithm’s deliberate seeking of gaps at every stage of a schedule’s creation, resulting in carefully stacked shipment unloading times. Of course, this method is aided by the uniformity of vehicle unload-times, which are in the algorithm artificially held constant, as a relationship between the size of vehicle, and the capacity for unloading that particular vehicle at that particular node. Nonetheless, the algorithm has finely used all available unloading space in this limited example.

A notable, potential error is the double-booking effect visible in the use of mode 2, node 3 unloading capacity. Shipment-components are being loaded simultaneously, resulting in unloading times taking up the same block, theoretically. This, of course, is an impossibility according to the strict rules of the problem. The shipments, according to our previous assumptions, use all unloading capacity totally and therefore cannot be unloaded simultaneously.

The cause of this apparent error is double-booking. The shipments are both attempting to use the latter half of the first day on which they both arrive, and the first half of the next day. As the algorithm does not have sufficient granularity to track half-days, the shipments are nominally double-booking.

The next table, Table 9, shows the data with a view to the particular assignments made to specific vehicles at specific times. Notably, it sorts by mode, then vehicle number, and then start-time, in order to demonstrate the relative efficiency and inefficiency of allocation according to the algorithm.

In this case, the algorithm generates an essentially perfect solution to the system, using virtually every open space, and the result is intuitively near-optimal. For mode 1, each vehicle can handle at most 3 assignments (as it requires three days to deliver any in particular, and the latest deadline is day 10). From this we know that, at least, we would need 5 vehicles, as we have 14 shipments. Shipment 7, our last shipment, has a deadline on day 7, and so it is apparent it could not easily be transferred to vehicle 4 or 5 with the shipment loads as they currently stand. While it is conceivable that a more efficient solution exists, the solution generated by the algorithm is intuitively near-optimal, at least for mode 1. On mode 2, we simply observe that both vehicles are identically full.

Table 9. Multimodal Simple Case Shipment Output

Mode	Vehicle	Start Time	Shipment	Sink
1	1	1	2	3
1	1	4	2	3
1	1	7	8	4
1	2	1	3	4
1	2	4	3	4
1	2	8	8	4
1	3	2	3	4
1	3	5	4	4
1	3	9	11	4
1	4	2	9	3
1	4	5	9	3
1	5	3	9	3
1	5	6	11	4
1	6	3	7	4
2	1	1	10	3
2	1	3	6	3
2	1	5	6	3
2	1	7	6	3
2	1	9	12	4
2	2	1	1	3
2	2	3	12	4
2	2	5	12	4
2	2	7	5	3
2	2	9	5	3

To compare to our earlier values, we have 6 vehicles of type 1, shipping 100 units every 3 days, and 2 vehicles of type 2, shipping 100 units every 2 days, for a total of 300 units of shipping capacity. This compares favorably to our lower bound estimate of 240 tons as an absolute minimum, given the effects of infrastructure interference and the uneven effect of making trips of duration 3 days during a space of 10 days.

Multimodal Large Scale Case

In order to test the algorithm's capacity to perform at large scale, a new shipping requirement list was created for the multimodal simple case. In this expansion of the original problem, we extrapolated the original data set to one hundred shipments, and extended the infrastructure capacity of the underlying network to handle 100,000 units per node per day, in order to ensure feasibility.

Solution of the problem required approximately five seconds. The data that was returned indicated that 212 vehicles were required to handle the shipping, all of the same mode, which approximately conforms to expectations. Given that infrastructure and deadline limitations were not concerns, all shipments should have been processed on the first mode. This was indeed the case.

Determining whether the solution was ultimately feasible would require detailed comparison of each shipment to each other shipment and to the overall infrastructure capacity and vehicle usage charts, in order to confirm the validity of the original result, but a superficial examination reveals start times which increase slowly as the infrastructure begins to fill.

A second attempt was made to process the same set of shipments, using infrastructure of only 100 per day. This resulted in massive over-flow, as predicted, over-flowing the limited 2-dimensional array for unloading. The array was defined to allow unloading to take place only twice the difference between the latest deadline and the earliest arrival from the earliest arrival. Because the shipping requirements were infeasible beyond anticipated infeasibility, the algorithm could not provide even an infeasible schedule.

V. Conclusions

Summary

The combination of Dijkstra's algorithm and scheduling heuristic developed here provide an efficient schedule for a multiple vehicle routing problem and an efficient high estimate on the number of vehicles required. In Chapter II, we showed that the field of solution techniques for large scale VRP is sparse when providing schedules for above 500 shipments. In Chapter IV, we demonstrated that the algorithm illustrated here can resolve the MVRP with 500 shipments in a very short period of time.

The algorithm is highly specialized. It requires that vehicles travel directly from source to sink, and then return to source. It also requires that vehicles be allowed to carry only one shipment or shipment component, that shipments be required to travel unimodally, and that speed and cost increase together continuously. Finally, it requires a large differential in cost between each vehicle type.

With those caveats, the algorithm is timely and effective. It provides efficient solutions in a short time frame at large scale. The solutions are heuristic but provide a feasible solution of good quality and a starting point for correction by Tabu search and other heuristics.

Future Research

The research suggests several notable avenues for future research. Firstly, the algorithm could be combined with another heuristic to generate more optimal solutions. For example, a Tabu search could be applied after the generation of the initial solution by this algorithm, providing a correction mechanism.

Secondly, the algorithm could be extended to true multi-modality for single shipments. By generating multimodal shortest paths, as well as unimodal shortest paths, in the Dijkstra's stage, and using an appropriate costing mechanism during the scheduling heuristic, the algorithm could handle multimodal paths, treating them as another type of vehicle.

Thirdly, the algorithm could be extended to handle multiple shipments on the same vehicle. The intuitive approach to this extension would be to add another step to the scheduling heuristic that attempted to use excess capacity on vehicles which were already scheduled. This would allow for more efficient use of the vehicles and better solutions. It would especially improve solutions in instances where the shipments were generally smaller than the capacity of a single vehicle.

Appendix A: Shortest Path Network Generator

This subprocedure is the core process for the Dijkstra’s algorithm component of the thesis. It iterates through the nodes, applies the Dijkstra’s subprocedure to them, and records the results of the Dijkstra’s algorithm for each source node in turn.

```
Sub simplifynetwork()
   Dim n As Integer
   Dim i As Integer
   Dim SourceNode As Integer
   Dim NetworkSheet As String
   Dim numnodes As Integer
   NetworkSheet = ActiveSheet.Name
   SourceNode = 1
   n = 1
   i = 1
   j = 1
   Sheets(NetworkSheet).Activate
   Range("A1").Select
   Do While ActiveCell.Offset(n, 0).Value <> 0
      n = n + 1
   Loop
      numnodes = n - 1
      Sheets.Add.Name = "Simplified " & NetworkSheet
      Sheets.Add.Name = "Simplified " & NetworkSheet & " Paths"
   Do While SourceNode <= numnodes
      DijkstrasAlgorithm SourceNode, numnodes, NetworkSheet
      SourceNode = SourceNode + 1
   Loop
   Sheets("Simplified " & NetworkSheet & " Paths").Activate
   ActiveCell.Value = "Nodes"
   Do While i <= numnodes
      ActiveCell.Offset(0, i) = Str(i)
      ActiveCell.Offset(i, 0) = Str(i)
      i = i + 1
   Loop
   Sheets("Simplified " & NetworkSheet).Activate
   ActiveCell.Value = "Nodes"
   Do While j <= numnodes
      ActiveCell.Offset(0, j) = Str(j)
      j = j + 1
```

```
    Loop
End Sub
```

This subprocedure is Dijkstra's Algorithm. It uses Dijkstra's method for defining the distance between a specific node and all other nodes in a network to define a list of distances between a source node and all other nodes.

```
Sub DijkstrasAlgorithm(SourceNode As Integer, numnodes As Integer, NetworkSheet As
String)
    Dim CurrentNode As Integer
    Dim CurrentDist As Double
    Dim AmDone As Boolean
    Dim ShortestDist() As Double
    Dim BigM As Double
    Dim n As Integer
    Dim i As Integer
    Dim j As Integer
    Dim CurrentLowDist As Double
    Dim CurrentLowNode As Integer
    Dim FoundUnexplored As Boolean
    Dim K As Integer
    Dim DistOnThisPath As Double
    Dim ShortestPath() As String
    Dim p As Integer
    AmDone = False
    CurrentNode = SourceNode
    CurrentDist = 0
    BigM = 1E+300
    n = 1
    ReDim ShortestDist(1 To numnodes)
    ReDim ShortestPath(1 To numnodes)
    Do While n <= numnodes
        ShortestDist(n) = BigM
        ShortestPath(n) = Str(SourceNode)
        n = n + 1
    Loop
    ShortestDist(SourceNode) = 0
    Do While AmDone = False
        i = 1
        Do While i <= numnodes
            DistOnThisPath = finddist(CurrentNode, i, NetworkSheet) +
ShortestDist(CurrentNode)
            If DistOnThisPath < ShortestDist(i) Then
              ShortestDist(i) = DistOnThisPath
              ShortestPath(i) = ShortestPath(CurrentNode) & Str(i)
            End If
            i = i + 1
        Loop
        j = 1
        CurrentLowDist = BigM
        FoundUnexplored = False
        Do While j <= numnodes
            If ShortestDist(j) > CurrentDist Then
              If ShortestDist(j) < CurrentLowDist Then
                 CurrentLowDist = ShortestDist(j)
                 CurrentLowNode = j
                 FoundUnexplored = True
              End If
            ElseIf ShortestDist(j) = CurrentDist Then
              If CurrentNode < j Then
                 CurrentLowDist = ShortestDist(j)
                 CurrentLowNode = j
                 FoundUnexplored = True
              End If
            End If
            j = j + 1
        Loop
        If FoundUnexplored = False Then
            AmDone = True
        ElseIf FoundUnexplored = True Then
            CurrentNode = CurrentLowNode
            CurrentDist = CurrentLowDist
        End If
    Loop
    Sheets("Simplified " & NetworkSheet).Activate
    ActiveSheet.Range("A1").Select
    K = 1
    Do While ActiveCell.Offset(K, 0).Value <> 0
        K = K + 1
    Loop
    ActiveCell.Offset(K, 0) = SourceNode
    l = 1
    Do While l < numnodes + 1
        ActiveCell.Offset(K, l) = ShortestDist(l)
        l = l + 1
    Loop
    Sheets("Simplified " & NetworkSheet & " Paths").Activate
    ActiveSheet.Range("A1").Select
```

```
    p = 1
    Do While p < numnodes + 1
        ActiveCell.Offset(K, p) = ShortestPath(p)
        p = p + 1
    Loop
End Sub
```

This function is used for finding the distances in the original network between any node and any other node, using direct paths. It is assumed that the original network contains connections between each path, although setting the distance arbitrarily large has the effect of eliminating the connection, since any shorter path will replace it.

```
Function finddist(SourceNode As Integer, sinknode As Integer, NetworkSheet As String)
    Sheets(NetworkSheet).Activate
    ActiveSheet.Range("A1").Select
    finddist = CDbl(ActiveCell.Offset(SourceNode, sinknode).Value)
End Function
```

Appendix B: Ordered Scheduling Heuristic

```
Public NumNodes As Integer
Public NumModes As Integer
Public numshipments As Integer
Public SimpleNetwork() As Variant
Public minvehicles() As Integer
Public latedeadline As Integer
Public highestused As Integer
Public ShArrival() As Variant
Public ShDeadline() As Variant
Public vehiclespeeds() As Variant
Public shsize As Variant
Public shsink As Variant
Public shsource As Variant
Public vcapacity As Variant
Public shUnloadTime() As Double
Public shipmentarray() As Integer
Public shipmentarraynum As Integer
Public vunloadtime() As Double
Public index As Integer
Public shtraveltime() As Double
```

This is the core procedure for the Heuristic Scheduler. It handles the process of tracking which mode is being minimized, as well as passing the array of shipments to be processed from component to component.

```
Sub HeuristicScheduler()
   Dim NumShipArray As Integer
   Dim DeadlineArray() As Integer
   Dim DeadlineTracker() As Integer
   Dim infraarray() As Integer
   Dim infratracker() As Integer
   Dim n As Integer
   Dim mode As Integer
   Popglobals
   PopulateSimpleNetwork
   index = 1
   ReDim shUnloadTime(1 To NumModes, 1 To numshipments)
   ReDim shipmentarray(1 To numshipments)
   shipmentarraynum = numshipments
```

```
    n = 1
        Do While n <= numshipments
        shipmentarray(n) = n
        n = n + 1
    Loop
    mode = NumModes
    Do While mode > 0
        DeadlineArrayGenerator shipmentarray(), shipmentarraynum, DeadlineArray(),
DeadlineTracker()
        InfraArrayGenerator DeadlineArray(), DeadlineTracker(), infraarray(), infratracker()
        ListVehicAssign infraarray(), infratracker(), mode
        eliminateshipments infraarray(), infratracker(), mode, shipmentarray(),
shipmentarraynum
        mode = mode - 1
    Loop
End Sub
```

This subprocedure is responsible for eliminating the shipments which have been assigned as part of the Vehicle Assignment step. It tracks through all assigned shipments, finds them in the original array, and deletes them, replacing them with the last value in the array, and reducing the size of the array by 1.

```
Sub eliminateshipments(infraarray() As Integer, infratracker() As Integer, mode As
Integer, shipmentarray() As Integer, shipmentarraynum As Integer)
   Dim n As Integer
   Dim k As Integer
   Dim elim As Boolean
   elim = False
   n = 1
   k = 1
   Do While n <= infratracker(mode)
      Do While k <= shipmentarraynum And elim = False
         If infraarray(mode, n) = shipmentarray(k) Then
            elim = True
            shipmentarray(k) = shipmentarray(shipmentarraynum)
            shipmentarraynum = shipmentarraynum - 1
         End If
         k = k + 1
      Loop
      elim = False
      k = 1
      n = n + 1
   Loop
End Sub
```

This subprocedure handles the process of choosing shipments from the output of the Infrastructure Assignment step to be assigned to specific vehicles. It iterates through the output of the Infrastructure Assignment step for the minimizing mode, in order of arrival date, and sends the shipments which are chosen to the shipment assignment step, ShipmentVehicAssign.

```
Sub ListVehicAssign(infraarray() As Integer, infratracker() As Integer, mode As Integer)
   Dim n As Integer
   Dim VehicAssignArray() As Double
   Dim unloadarray() As Double
   Dim currmode As Integer
   ReDim VehicAssignArray(1 To minvehicles(mode), 1 To latedeadline)
   ReDim unloadarray(1 To NumNodes, 1 To latedeadline)
   highestused = 0
   n = 1
   Do While n <= infratracker(mode)
      ShipmentVehicAssign VehicAssignArray(), infraarray(mode, n), mode,
unloadarray(), highestused
      n = n + 1
   Loop
   minvehicles(mode) = highestused
   If mode > 1 Then
      listgapcheck VehicAssignArray(), infraarray(), infratracker(mode - 1), mode,
unloadarray()
   End If
End Sub
```

ListGapCheck iterates through the reduced shipment list, after the Vehicle Assignment step, and sends each shipment to ShipmentGapCheck. ShipmentGapCheck then determines whether the current infrastructure and vehicle assignments for the minimized mode can handle the additional shipment. ShipmentGapCheck then eliminates the shipment, if it can be assigned, and fills in the appropriate gaps.

```
Sub listgapcheck(VehicAssignArray() As Double, ShipmentList() As Integer, infranum
As Integer, currmode As Integer, unloadarray() As Double)
   Dim n As Integer
   Dim found As Boolean
   Dim currvehicle As Integer
   n = 1
   Do While n <= infranum
      ShipmentGapCheck VehicAssignArray(), ShipmentList(), n, infranum, currmode,
unloadarray()
      n = n + 1
   Loop
End Sub
```

ShipmentGapCheck attempts to assign the shipment given to it to the recently minimized mode, searching for any gaps large enough to carry the current shipment. If it can find such a gap, it fills the gap, and deletes the shipment from the array of unassigned shipments.

```
Sub ShipmentGapCheck(VehicAssignArray() As Double, ShipmentList() As Integer,
ShipmentNumber As Integer, infranum As Integer, mode As Integer, unloadarray() As
Double)
   Dim currvehicle As Integer
   Dim found As Boolean
   Dim currstart As Double
   Dim starttimes() As Double
   Dim foundvehicles As Integer
   Dim vehicles() As Integer
   Dim shipment As Integer
   Dim i As Integer
   Dim j As Integer
   Dim elim As Boolean
   Dim pseudounloadarray() As Double
   Dim l As Integer
   Dim k As Integer
   Dim traveltime As Double
   ReDim pseudounloadarray(1 To NumNodes, 1 To latedeadline)
   shipment = ShipmentList(mode - 1, ShipmentNumber)
   traveltime = SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) /
vehiclespeeds(mode)
   currvehicle = 1
   l = 1
   k = 1
   Do While l <= NumNodes
      Do While k <= latedeadline
         pseudounloadarray(l, k) = unloadarray(l, k)
         k = k + 1
      Loop
      k = 1
      l = l + 1
   Loop
   If shsize(shipment) / vcapacity(mode) - Int(shsize(shipment) / vcapacity(mode)) > 0
Then
      neededvehicles = Int(shsize(shipment) / vcapacity(mode)) + 1
   Else
      neededvehicles = Int(shsize(shipment) / vcapacity(mode))
   End If
   ReDim vehicles(1 To neededvehicles)
```

```
    ReDim starttimes(1 To neededvehicles)
    Do While currvehicle <= minvehicles(mode) And foundvehicles < neededvehicles
        If foundvehicles > 0 Then
            If vehicles(foundvehicles) = currvehicle Then
                currstart = starttimes(foundvehicles) + vunloadtime(mode, shsink(shipment)) +
2 * SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) / vehiclespeeds(mode)
            Else
                currstart = 0
            End If
        Else
            currstart = 0
        End If
        found = True
        corrected = True
        Do While found = True And corrected = True
            found = findstarttime(VehicAssignArray(), mode, starttimes(foundvehicles + 1),
currvehicle, currstart, shipment)
            corrected = correctstarttime(pseudounloadarray(), mode, starttimes(foundvehicles
+ 1), currvehicle, found, shipment, currstart)
        Loop
        If found = True Then
            foundvehicles = foundvehicles + 1
            allocate starttimes(foundvehicles) + traveltime, pseudounloadarray(),
shsink(shipment), vunloadtime(mode, shsink(shipment))
            vehicles(foundvehicles) = currvehicle
            n = n + 1
        Else
            currvehicle = currvehicle + 1
            corrected = True
            found = True
        End If
    Loop
    i = 1
    If foundvehicles >= neededvehicles Then
        Do While i <= neededvehicles
            allocshipvehic starttimes(i), vehicles(i), shipment, VehicAssignArray(),
unloadarray(), mode
            i = i + 1
        Loop
        j = 1
        Do While j <= infranum And elim = False
            If shipmentarray(j) = shipment Then
                shipmentarray(j) = shipmentarray(shipmentarraynum)
                shipmentarraynum = shipmentarraynum - 1
                elim = True
            End If
            j = j + 1
         Loop
      End If
   End Sub
```

ShipmentVehicAssign handles the process of tracking the starttimes for various components of a shipment, and passes the components to the parts of the algorithm to findstarttime and correctstarttime, the subprocedures responsible for finding gaps large enough to handle a shipment, both in the vehicle array, and in the unloading array.

It also maintains the pseudo unload array, an array used to track hypothetical points of unloading for shipment components prior to their final assignment.

```
Sub ShipmentVehicAssign(VehicAssignArray() As Double, shipment As Integer, mode
As Integer, unloadarray() As Double, highestused As Integer)
   Dim neededvehicles As Integer
   Dim n As Integer
   Dim i As Integer
   Dim starttimes() As Double
   Dim found As Boolean
   Dim corrected As Boolean
   Dim currstart As Double
   Dim vehicles() As Integer
   Dim currvehicle As Integer
   Dim pseudounloadarray() As Double
   Dim j As Integer
   Dim k As Integer
   Dim temphighestused As Integer
   Dim traveltime As Double
   ReDim pseudounloadarray(1 To NumNodes, 1 To latedeadline)
   traveltime = SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) /
vehiclespeeds(mode)
   j = 1
   k = 1
   Do While j <= NumNodes
      Do While k <= latedeadline
         pseudounloadarray(j, k) = unloadarray(j, k)
         k = k + 1
      Loop
      k = 1
      j = j + 1
   Loop
   If shsize(shipment) / vcapacity(mode) - Int(shsize(shipment) / vcapacity(mode)) > 0
Then
      neededvehicles = Int(shsize(shipment) / vcapacity(mode)) + 1
   Else
      neededvehicles = Int(shsize(shipment) / vcapacity(mode))
   End If
   ReDim starttimes(1 To neededvehicles)
   ReDim vehicles(1 To neededvehicles)
   n = 1
```

```
    currvehicle = 1
    temphighestused = highestused
    Do While n <= neededvehicles
        corrected = True
        found = True
        If foundvehicles > 0 Then
            If vehicles(foundvehicles) = currvehicle Then
                currstart = starttimes(foundvehicles) + vunloadtime(mode, shsink(shipment)) +
2 * SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) / vehiclespeeds(mode)
            Else
                currstart = 0
            End If
        Else
            currstart = 0
        End If
        Do While corrected = True And found = True
            found = findstarttime(VehicAssignArray(), mode, starttimes(n), currvehicle,
currstart, shipment)
            corrected = correctstarttime(pseudounloadarray(), mode, starttimes(n),
currvehicle, found, shipment, currstart)
        Loop
        If found = True Or currvehicle > temphighestused Then
            vehicles(n) = currvehicle
            foundvehicles = foundvehicles + 1
            allocate starttimes(n) + traveltime, pseudounloadarray(), shsink(shipment),
vunloadtime(mode, shsink(shipment))
            If vehicles(foundvehicles) > temphighestused Then
                temphighestused = vehicles(foundvehicles)
            End If
            n = n + 1
        Else
            currvehicle = currvehicle + 1
            currstart = 0
            corrected = True
            found = True
        End If
    Loop
    i = 1
    Do While i <= neededvehicles
        If shipment = 3 Then
            shipment = shipment
        End If
        allocshipvehic starttimes(i), vehicles(i), shipment, VehicAssignArray(),
unloadarray(), mode
        i = i + 1
    Loop
    If vehicles(neededvehicles) > highestused Then
        highestused = vehicles(neededvehicles)
    End If
End Sub
```

This procedure is used to allocate a shipment component whose starttime is known. The algorithm simply fills in the gap it is passed, using the shipment and vehicle information to determine how much of the gap must be filled in.

```
Sub allocshipvehic(starttime As Double, vehicle As Integer, shipment As Integer,
VehicAssignArray() As Double, unloadarray() As Double, mode As Integer)
   Dim reqvehictime As Double
   Dim requnloadtime As Double
   Dim allocvehictime As Double
   Dim allocunloadtime As Double
   Dim currtime As Double
   Dim timetransfervariable As Double
   Dim currday As Integer
   Sheets("Output").Range("A1").Offset(index, 0) = shipment
   Sheets("Output").Range("A1").Offset(index, 1) = vehicle
   Sheets("Output").Range("A1").Offset(index, 2) = starttime
   Sheets("Output").Range("A1").Offset(index, 3) = mode
   Sheets("Output").Range("A1").Offset(index, 4) = shsink(shipment)
   index = index + 1
   reqvehictime = SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) /
vehiclespeeds(mode) * 2 + vunloadtime(mode, shsink(shipment))
   requnloadtime = vunloadtime(mode, shsink(shipment))
   currday = Int(starttime)
   allocvehictime = 0
   Do While allocvehictime < reqvehictime
      timetransfervariable = 1
      If timetransfervariable > (1 - starttime + currday) Then
         timetransfervariable = 1 - starttime + currday
      End If
      If timetransfervariable > 1 - VehicAssignArray(vehicle, currday) Then
         timetransfervariable = 1 - VehicAssignArray(vehicle, currday)
      End If
      If timetransfervariable > reqvehictime - allocvehictime Then
         timetransfervariable = reqvehictime - allocvehictime
      End If
      VehicAssignArray(vehicle, currday) = VehicAssignArray(vehicle, currday) +
timetransfervariable
      allocvehictime = allocvehictime + timetransfervariable
      currday = currday + 1
   Loop
   currday = Int(starttime + SimpleNetwork(mode)(shsource(shipment),
shsink(shipment)) / vehiclespeeds(mode))
   Do While allocunloadtime < requnloadtime
      timetransfervariable = requnloadtime - allocunloadtime
```

```
        If timetransfervariable > (1 - starttime - SimpleNetwork(mode)(shsource(shipment),
shsink(shipment)) / vehiclespeeds(mode) + currday) Then
            timetransfervariable = (1 - starttime - SimpleNetwork(mode)(shsource(shipment),
shsink(shipment)) / vehiclespeeds(mode) + currday)
        End If
        If timetransfervariable > 1 - unloadarray(shsink(shipment), currday) Then
            timetransfervariable = 1 - unloadarray(shsink(shipment), currday)
        End If
        unloadarray(shsink(shipment), currday) = unloadarray(shsink(shipment), currday) +
timetransfervariable
        allocunloadtime = allocunloadtime + timetransfervariable
        currday = currday + 1
    Loop
End Sub
```

This function is used to determine whether a certain start time found in the vehicle array to be large enough for a particular shipment component has a corresponding gap in the unload array which is large enough to handle the shipment. If not, the function finds the next gap that is large enough, and returns it as a suggestion to the vehicle search component.

```
Function correctstarttime(infraassignarray() As Double, mode As Integer, starttime As
Double, currvehicle As Integer, found As Boolean, shipment As Integer, currstart As
Double)
   Dim currunloadtime As Double
   Dim currday As Integer
   Dim traveltime As Double
   Dim availtime As Double
   Dim curravailtime As Double
   traveltime = SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) /
vehiclespeeds(mode)
   currunloadtime = starttime + traveltime
   currday = Int(currunloadtime)
   Do While availtime < vunloadtime(mode, shsink(shipment)) And currday <
latedeadline
      curravailtime = 1
      If 1 - infraassignarray(shsink(shipment), currday) < curravailtime Then
         curravailtime = 1 - infraassignarray(shsink(shipment), currday)
      End If
      If 1 - (currunloadtime - currday) < curravailtime Then
         curravailtime = 1 - (currunloadtime - currday)
      End If
      availtime = availtime + curravailtime
      If infraassignarray(shsink(shipment), currday) > 0 And availtime <
vunloadtime(mode, shsink(shipment)) Then
         currunloadtime = currday + infraassignarray(shsink(shipment), currday)
         currday = currday + 1
         availtime = currday - currunloadtime
      End If
   Loop
   If currunloadtime = starttime + traveltime Then
      correctstarttime = False
   Else
      correctstarttime = True
      currstart = currunloadtime - traveltime
   End If
   If availtime >= vunloadtime(mode, shsink(shipment)) And currunloadtime +
vunloadtime(mode, shsink(shipment)) <= ShDeadline(shipment) + 1 Then
      found = True
   Else
```

```
        found = False
    End If
End Function
```

This function searches for the first gap in the vehicle array, for the current vehicle, capable of carrying a given shipment component. If found, it returns the time at which the gap is found.

```
Function findstarttime(VehicAssignArray() As Double, mode As Integer, starttime As
Double, currvehicle As Integer, currstart As Double, shipment As Integer)
   Dim currtime As Double
   Dim currday As Integer
   Dim reqtime As Double
   Dim availtime As Double
   Dim curravailtime As Double
   If ShArrival(shipment) > currstart Then
      currtime = ShArrival(shipment)
   Else
      currtime = currstart
   End If
   currday = Int(currtime)
   reqtime = SimpleNetwork(mode)(shsource(shipment), shsink(shipment)) /
vehiclespeeds(mode) * 2 + vunloadtime(mode, shsink(shipment))
   Do While availtime < reqtime And currday < latedeadline
      curravailtime = 1
      If 1 - VehicAssignArray(currvehicle, currday) < curravailtime Then
         curravailtime = 1 - VehicAssignArray(currvehicle, currday)
      End If
      If 1 - (currtime - currday) < curravailtime Then
         curravailtime = 1 - (currtime - currday)
      End If
      availtime = availtime + curravailtime
      If VehicAssignArray(currvehicle, currday) > 0 And availtime < reqtime Then
         availtime = 1 - VehicAssignArray(currvehicle, currday)
         currtime = currday + VehicAssignArray(currvehicle, currday)
      End If
      currday = currday + 1
   Loop
   starttime = currtime
   If availtime >= reqtime And starttime + SimpleNetwork(mode)(shsource(shipment),
shsink(shipment)) / vehiclespeeds(mode) + vunloadtime(mode, shsink(shipment)) <
ShDeadline(shipment) + 1 Then
      findstarttime = True
   Else
      findstarttime = False
   End If
End Function
```

This procedure handles the top level of processing for the infrastructure assignment step. It passes the mode to be assigned to the infraassignstep, which then performs infrastructure assignment for the specific mode.

```
Sub InfraArrayGenerator(DeadlineArray() As Integer, DeadlineTracker() As Integer,
infraarray() As Integer, infratracker() As Integer)
    Dim n As Integer
    ReDim infraarray(1 To NumModes, 1 To numshipments)
    ReDim infratracker(1 To NumModes)
    n = 1
    Do While n <= NumModes
        InfraAssignStep DeadlineArray(), DeadlineTracker(), infraarray(), infratracker(), n
        n = n + 1
    Loop
End Sub
```

This subprocedure handles the process of allocating the shipments to unloading capacity. If the shipment can be allocated, it is retained for this mode. If no gap can be found, the algorithm assigns it upwards to the next mode.

```
Sub InfraAssignStep(DeadlineArray() As Integer, DeadlineTracker() As Integer,
infraarray() As Integer, infratracker() As Integer, currmode As Integer)
   Dim infraassignarray() As Double
   Dim n As Integer
   Dim currarrival As Integer
   Dim i As Integer
   Dim shipment As Integer
   ReDim shtraveltime(1 To numshipments)
   ShDeadline =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("F2",
Sheets("Shipments").Range("F2").End(xlDown)))
   ShArrival =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("E2",
Sheets("Shipments").Range("E2").End(xlDown)))
   shsink = Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("C2",
Sheets("Shipments").Range("C2").End(xlDown)))
   populatetraveltime currmode, shtraveltime(), shsink
   n = 1
   latedeadline = 0
   Do While n <= numshipments
      If ShDeadline(n) > latedeadline Then
         latedeadline = ShDeadline(n)
      End If
      If ShArrival(n) > latearrival Then
         latearrival = ShArrival(n)
      End If
      n = n + 1
   Loop
   latedeadline = latedeadline * 2
   ReDim infraassignarray(1 To NumNodes, 1 To latedeadline)
   i = 1
   currarrival = 1
   Do While currarrival <= latearrival
      Do While i <= DeadlineTracker(currmode)
         shipment = DeadlineArray(currmode, i)
         If currarrival = ShArrival(shipment) Then
            found = findspace(shipment, infraassignarray(), shtraveltime(shipment) +
ShArrival(shipment), ShDeadline(shipment), shsink(shipment),
shUnloadTime(currmode, shipment))
            If found = True Or currmode = NumModes Then
```

```
                    allocate ShArrival(shipment) + shtraveltime(shipment), infraassignarray(),
shsink(shipment), shUnloadTime(currmode, shipment)
                    infratracker(currmode) = infratracker(currmode) + 1
                    infraarray(currmode, infratracker(currmode)) = DeadlineArray(currmode, i)
                    found = False
                Else
                    DeadlineTracker(currmode + 1) = DeadlineTracker(currmode + 1) + 1
                    DeadlineArray(currmode + 1, DeadlineTracker(currmode + 1)) =
DeadlineArray(currmode, i)
                End If
            End If
            i = i + 1
        Loop
        i = 1

        currarrival = currarrival + 1
    Loop
End Sub
```

Calculates the traveltime required for each shipment, for each mode.

```
Sub populatetraveltime(mode As Integer, traveltime() As Double, sink As Variant)
    Dim n As Integer
    vehiclespeeds =
Application.WorksheetFunction.Transpose(Sheets("Vehicles").Range("B2",
Sheets("Vehicles").Range("B2").End(xlDown)))
    shsource =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("B2",
Sheets("Shipments").Range("B2").End(xlDown)))
    n = 1
    Do While n <= numshipments
        traveltime(n) = SimpleNetwork(mode)(shsource(n), sink(n)) / vehiclespeeds(mode)
        n = n + 1
    Loop
    n = 1
End Sub
```

Calculates the amount of time that a shipment will require on the unload array in order to be totally unloaded.

```
Sub populateneededtime(mode As Integer, sink As Variant)
   Dim n As Integer
   Dim infraarray() As Variant
   Dim transfervariable As Variant
   Dim j As Integer
   Dim k As Integer
   ReDim infraarray(1 To NumModes)
   ReDim vunloadtime(1 To NumModes, 1 To NumNodes)
   shsize = Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("D2",
Sheets("Shipments").Range("D2").End(xlDown)))
   j = 1
   Do While j <= NumModes
      With Sheets("Network " & j & " Infra").Range("B2")
         transfervariable = Application.WorksheetFunction.Transpose(Range(.Offset(0, 0),
.End(xlDown)))
      End With
      infraarray(j) = transfervariable
      k = 1
      Do While k <= NumNodes
         vunloadtime(j, k) = vcapacity(mode) / infraarray(j)(k)
         k = k + 1
      Loop
      j = j + 1
   Loop
   n = 1
   Do While n <= numshipments
      shUnloadTime(mode, n) = shsize(n) / infraarray(mode)(sink(n))
      n = n + 1
   Loop
End Sub
```

Checks the current infrastructure array to find gaps to which a shipment could be assigned. If it cannot find a gap, returns as false. Otherwise, returns as true.

```
Function findspace(shipment As Integer, infraassignarray() As Double, startpoint As
Double, deadline As Variant, sink As Variant, neededtime As Double)
   Dim foundtime As Double
   Dim currday As Integer
   currday = Int(startpoint)
   Do While foundtime < neededtime And currday <= deadline
      foundtime = 1 - infraassignarray(sink, currday) + foundtime
      If 1 - (startpoint - currday) < foundtime Then
         foundtime = 1 - (startpoint - currday)
      End If
      currday = currday + 1
   Loop
   If foundtime >= neededtime Then
      findspace = True
   Else
      findspace = False
   End If
End Function
```

If findspace has found a gap during the infrastructure assignment phase, then the allocate component will fill in the gap, ensuring against double-booking.

```
Sub allocate(startpoint As Double, infraassignarray() As Double, sink As Variant,
neededtime As Double)
   Dim remainingtime As Double
   Dim currday As Integer
   Dim transfertime As Double
   currday = Int(startpoint)
   remainingtime = neededtime
   Do While remainingtime > 0
      transfertime = remainingtime
      If transfertime > 1 - infraassignarray(sink, currday) Then
         transfertime = 1 - infraassignarray(sink, currday)
      End If
      If transfertime > currday + 1 - startpoint Then
         transfertime = currday + 1 - startpoint
      End If
      infraassignarray(sink, currday) = infraassignarray(sink, currday) + transfertime
      remainingtime = remainingtime - transfertime
      currday = currday + 1
   Loop
End Sub
```

This subprocedure populates the majority of the global variables used in later calculations from the excel sheets used as input.

```
Sub Popglobals()
   Dim vehicleestimate As Integer
   Dim i As Integer
   Dim mode As Integer
   numshipments = Sheets("Shipments").Range("A1",
Sheets("Shipments").Range("A1").End(xlDown)).Rows.Count - 1
   NumModes = Sheets("Vehicles").Range("A1",
Sheets("Vehicles").Range("A1").End(xlDown)).Rows.Count - 1
   NumNodes = Sheets("Simplified Network 1").Range("A1", Sheets("Simplified
Network 1").Range("A1").End(xlDown)).Rows.Count - 1
   shsize = Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("D2",
Sheets("Shipments").Range("D2").End(xlDown)))
   vcapacity =
Application.WorksheetFunction.Transpose(Sheets("Vehicles").Range("C2",
Sheets("Vehicles").Range("C2").End(xlDown)))
   vehicleestimate = 0
   i = 1
   mode = 1
   ReDim minvehicles(1 To NumModes)
   Do While mode <= NumModes
      Do While i <= numshipments
            If shsize(i) / vcapacity(mode) > 1 Then
            vehicleestimate = vehicleestimate + shsize(i) / vcapacity(mode)
         Else
            vehicleestimate = vehicleestimate + 1
         End If
         i = i + 1
      Loop
      i = 1
      minvehicles(mode) = Int(vehicleestimate * 2)
      mode = mode + 1
      vehicleestimate = 0
   Loop
End Sub
```

This procedure handles the assignment of shipments to individual modes based upon their feasibility. DetFeas generates an array of modes, called FeasArray. Each shipment's serial corresponds to its cheapest feasible mode, in FeasArray. The shipments are then assigned to the array corresponding to their cheapest feasible mode.

```
Sub DeadlineArrayGenerator(shipmentarray() As Integer, NumShipArray As Integer,
DeadlineArray() As Integer, DeadlineTracker() As Integer)
   Dim n As Integer
   Dim mode As Integer
   Dim FeasArray() As Integer
   ReDim DeadlineArray(1 To NumModes, 1 To NumShipArray)
   ReDim DeadlineTracker(1 To NumModes)
   ReDim FeasArray(1 To numshipments)
   n = 1
   DetFeas FeasArray()
   Do While n <= NumShipArray
      mode = FeasArray(shipmentarray(n))
      DeadlineTracker(mode) = DeadlineTracker(mode) + 1
      DeadlineArray(mode, DeadlineTracker(mode)) = shipmentarray(n)
      n = n + 1
   Loop
End Sub
```

This procedure populates FeasArray so that FeasArray(Shipment Number) will return the cheapest feasible mode for that shipment.

```
Sub DetFeas(FeasArray() As Integer)
   Dim shsource As Variant
   Dim shsink As Variant
   Dim ShArrival As Variant
   Dim ShDeadline As Variant
   Dim vehiclespeeds As Variant
   Dim found As Boolean
   Dim n As Integer
   Dim k As Integer
   Dim i As Integer
   ReDim FeasArray(1 To numshipments)
   shsource =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("B2",
Sheets("Shipments").Range("B2").End(xlDown)))
   shsink = Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("C2",
Sheets("Shipments").Range("C2").End(xlDown)))
   ShArrival =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("E2",
Sheets("Shipments").Range("E2").End(xlDown)))
   ShDeadline =
Application.WorksheetFunction.Transpose(Sheets("Shipments").Range("F2",
Sheets("Shipments").Range("F2").End(xlDown)))
   vehiclespeeds =
Application.WorksheetFunction.Transpose(Sheets("Vehicles").Range("B2",
Sheets("Vehicles").Range("B2").End(xlDown)))
   i = 1
   Do While i <= NumModes
      populateneededtime i, shsink
      i = i + 1
   Loop
   n = 1
   k = 1
   Do While n <= numshipments
      Do While k < NumModes And found = False
         If SimpleNetwork(k)(shsource(n), shsink(n)) / vehiclespeeds(k) +
shUnloadTime(k, n) <= ShDeadline(n) - ShArrival(n) + 1 Then
            found = True
            FeasArray(n) = k
         End If
         k = k + 1
      Loop
      If found = False Then
```

```
            FeasArray(n) = k
        End If
        found = False
        k = 1
        n = n + 1
    Loop
End Sub
```

It is assumed that Dijkstra's will only be run once, while the Heuristic Scheduler may be run many times. As a consequence, the simplification of the network outputs to Excel. This algorithm converts the output of the Dijkstra's component into arrays, for the Heuristic Scheduler to use internally.

```
Sub PopulateSimpleNetwork() 'This converts the simple networks generated by Dijkstras
algorithm into arrays, for speed.
   Dim i As Integer
   Dim transfervariable() As Variant
   ReDim SimpleNetwork(1 To NumModes)
   i = 1
   Do While i <= NumModes
      With Sheets("Simplified Network " & i).Range("B2")
         transfervariable = Range(.Offset(0, 0), .End(xlDown).End(xlToRight))
      End With
      SimpleNetwork(i) = transfervariable()
      i = i + 1
   Loop
End Sub
```

Bibliography

Dantzig, G. B., and J. H. Ramser. (1959). "The Truck Dispatching Problem." *Management Science* 6.1: 80-91. Print.

LaPorte, Gilbert. (2009). "Fifty Years of Vehicle Routing." *Transportation Science* 43.4: 408-16. Print.

Lu, Quan, and Maged Dessouky. (2004). "An Exact Algorithm for the Multiple Vehicle Pickup and Delivery Problem." *Transportation Science* 38.4: 503-14. Print.

Moccia, Luigi, Jean-Francois Cordeau, Gilbert Laporte, Stefan Ropke, and Maria Valentini. *Modeling and Solving a Multimodal Routing Problem With Timetables and Time Windows*. Tech. Dipartimento Di Elettronica, Informatica e Sistemistica, Università Della Calabria, n.d. Web. 20 Nov. 2012. <http://www.diku.dk/~sropke/>.

Psaraftis, H. N. (1983). "An Exact Algorithm for the Single Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows." *Transportation Science* 17.3: 351-57. Print.

Psaraftis, Harilaos. (1980). "A Dynamic Programming Approach to the Single-Vehicle, Many-to-Many Immediate Request Dial-a-Ride Problem." *Transportation Science* 14: 130-54. Print.

Savelsbergh, M. W. P., and M. Sol. (1995). "The General Pickup and Delivery Problem." *Transportation Science* 29.1: 17-29. Print.

Sexton, T. R., and L. D. Bodin. (1985). "Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: I. Scheduling." *Transportation Science* 19.4: 378-435. Print.

Verma, Sushil, and Maged Dessouky. (1999). "Multistage Hybrid Flowshop Scheduling with Identical Jobs and Uniform Parallel Machines." *Journal of Scheduling* 2.3: 135-50. Print.

Wagner, Harvey M. (1959). "An Integer Linear-programming Model for Machine Scheduling." *Naval Research Logistics Quarterly* 6.2: 131-40. Print.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704–0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.			
1. REPORT DATE (DD–MM–YYYY) 21-03-2013	2. REPORT TYPE Master's Thesis		3. DATES COVERED *(From — To)* September 2011-February 2013
4. TITLE AND SUBTITLE Vehicle Minimization for the Multimodal Pickup and Delivery Problem with Time Windows			5a. CONTRACT NUMBER
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Clapp, Benjamin A.			5d. PROJECT NUMBER 13S141
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT) 2950 Hobson Way WPAFB OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER ENS 13-M-03
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Amy Pappas, CIV USTRANSCOM/TCAC 508 Scott Drive Scott AFB IL 62225-5357 amy.a.pappas.civ@mail.mil 618-220-7758 DSN 770-7758			10. SPONSOR/MONITOR'S ACRONYM(S) TCAC
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.			

14. ABSTRACT

The algorithm proposed here is used for heuristic solutions for the Multimodal Multiple Vehicle Routing Problem with Unloading Capacity, Pickup and Dropoff, and Time Windows, solved so as to minimize the number of vehicles used, subject to varying objective function values for each vehicle. The MVRP is simplified and split into a routing problem and a scheduling problem.

The routing problem is addressed by Dijkstra's Algorithm. This generates a new network for the second stage of the algorithm. It is assumed that the shortest path is the correct path to use, and shipments each travel unimodally.

The scheduling problem is addressed by treating the various paths as though they were machines, with vehicle number being treated approximately as capacity for the machines, and unloading capacity being treated as a second stage in the processing. The problem is analyzed by assigning all shipments which can be assigned elsewhere away from the most expensive mode and then assigning only leftover shipments to the most expensive mode.

Multiple resolutions of the scheduling problem result in feasible solutions for less expensive modes, which results in a feasible solution for every mode, and a low cost solution in terms of vehicles used.

15. SUBJECT TERMS
Transportation Routing Heuristic VRP FSMP

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Dr. Jeffery Weir ENS
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	86	19b. TELEPHONE NUMBER (Include Area Code) (937)255-3636, ext 4523

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18